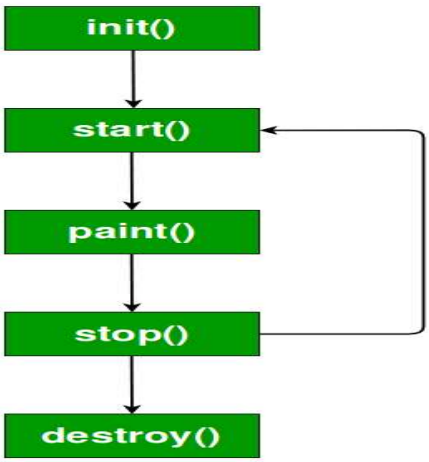
RTM Nagpur University Summer 2017
Solutions Set
Java Programming

1. How to create java applets. Explain in detail?

Java Applet Basics
Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.
Applets are used to make the web site more dynamic and entertaining.
Some important points :
All applets are sub-classes (either directly or indirectly) of java.applet.Applet class.
Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
In general, execution of an applet does not begin at main() method.
Output of an applet window is not performed by System.out.println(). Rather it is handled with various AWT methods, such as drawString().
Life cycle of an applet :



It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:
1. init( )
2. start( )
3. paint( )
When an applet is terminated, the following sequence of method calls takes place:
1. stop( )
2. destroy( )
Let's look more closely at these methods.
init( ) : The init( ) method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.
start( ) : The start( ) method is called after init( ). It is also called to restart an applet after it has been stopped. Note that init( ) is called once i.e. when the first time an applet is loaded whereas

start( ) is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( ).
paint( ) : The paint( ) method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.
paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called.
The paint( ) method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.
stop( ) : The stop( ) method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop( ) is called, the applet is probably running. You should use stop( ) to suspend threads that don't need to run when the applet is not visible. You can restart them when start( ) is called if the user returns to the page.
destroy( ) : The destroy( ) method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The stop( ) method is always called before destroy( ).
Creating Hello World applet :
Let's begin with the HelloWorld applet :
filter_none
edit
play_arrow
brightness_4

```
// A Hello World Applet
// Save file as HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
   // Overriding paint() method
   @Override
   public void paint(Graphics g)
   {
      g.drawString("Hello World", 20, 20);
   }

}
```

Explanation :
The above java program begins with two import statements. The first import statement imports the Applet class from applet package. Every AWT-based(Abstract Window Toolkit) applet that

you create must be a subclass (either directly or indirectly) of Applet class. The second statement import the Graphics class from awt package.
The next line in the program declares the class HelloWorld. This class must be declared as public, because it will be accessed by code that is outside the program. Inside HelloWorld, paint( ) is declared. This method is defined by the AWT and must be overridden by the applet.
Inside paint( ) is a call to drawString( ), which is a member of the Graphics class. This method outputs a string beginning at the specified X,Y location. It has the following general form:
void drawString(String message, int x, int y)
Here, message is the string to be output beginning at x,y. In a Java window, the upper-left corner is location 0,0. The call to drawString( ) in the applet causes the message "Hello World" to be displayed beginning at location 20,20.
Notice that the applet does not have a main( ) method. Unlike Java programs, applets do not begin execution at main( ). In fact, most applets don't even have a main( ) method. Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.
Running the HelloWorld Applet :
After you enter the source code for HelloWorld.java, compile in the same way that you have been compiling java programs(using javac command). However running HelloWorld with the java command will generate an error because it is not an application.

java HelloWorld

Error: Main method not found in class HelloWorld, please define the main method as:
  public static void main(String[] args)
There are two standard ways in which you can run an applet :
Executing the applet within a Java-compatible web browser.
Using an applet viewer, such as the standard tool, appletviewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.
Each of these methods is described next.
Using java enabled web browser : To execute an applet in a web browser we have to write a short HTML text file that contains a tag that loads the applet. We can use APPLET or OBJECT tag for this purpose. Using APPLET, here is the HTML file that executes HelloWorld :
<applet code="HelloWorld" width=200 height=60>
</applet>
The width and height statements specify the dimensions of the display area used by the applet. The APPLET tag contains several other options. After you create this html file, you can use it to execute the applet.
NOTE : Chrome and Firefox no longer supports NPAPI (technology required for Java applets). Refer here
Using appletviewer : This is the easiest way to run an applet. To execute HelloWorld with an applet viewer, you may also execute the HTML file shown earlier. For example, if the preceding HTML file is saved with
RunHelloWorld.html,then the following command line will run HelloWorld :
appletviewer RunHelloWorld.html

appletviewer with java source file : If you include a comment at the head of your Java source code file that contains the APPLET tag then your code is documented with a prototype of the necessary HTML statements, and you can run your compiled applet merely by starting the applet viewer with your Java source code file. If you use this method, the HelloWorld source file looks like this :

filter_none
edit
play_arrow
brightness_4

```
// A Hello World Applet
// Save file as HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;

/*
<applet code="HelloWorld" width=200 height=60>
</applet>
*/

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
   // Overriding paint() method
   @Override
   public void paint(Graphics g)
   {
      g.drawString("Hello World", 20, 20);
   }

}
```

With this approach, first compile HelloWorld.java file and then simply run below command to run applet :

appletviewer HelloWorld

Features of Applets over HTML

Displaying dynamic web pages of a web application.
Playing sound files.
Displaying documents
Playing animations

This subtopic is contributed by Surya Priy.

Restrictions imposed on Java applets

Due to security reasons , the following restrictions are imposed on Java applets:

1. An applet cannot load libraries or define native methods.

2. An applet cannot ordinarily read or write files on the execution host.
3. An applet cannot read certain system properties.
4. An applet cannot make network connections except to the host that it came from.
5. An applet cannot start any program on the host that's executing it.

2. Explain the two ways to create a class that can be distinguished

Classes and Objects in Java
Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

Modifiers : A class can be public or has default access (Refer this for details).
Class name: The name should begin with a initial letter (capitalized by convention).
Superclass(if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
Interfaces(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
Body: The class body surrounded by braces, { }.
Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

There are various types of classes that are used in real time applications such as nested classes, anonymous classes, lambda expressions.

Object

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

State : It is represented by attributes of an object. It also reflects the properties of an object.
Behavior : It is represented by methods of an object. It also reflects the response of an object with other objects.
Identity : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object : dog

Blank Diagram - Page 1 (5)

Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example :

Blank Diagram - Page 1 (3)

As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general,we can’t create objects of an abstract class or an interface.

Dog tuffy;
If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

filter_none
edit
play_arrow

brightness_4
```
// Class Declaration

public class Dog
{
   // Instance Variables
   String name;
```

```
String breed;
int age;
String color;

// Constructor Declaration of Class
public Dog(String name, String breed,
            int age, String color)
{
   this.name = name;
   this.breed = breed;
   this.age = age;
   this.color = color;
}

// method 1
public String getName()
{
   return name;
}

// method 2
public String getBreed()
{
   return breed;
}

// method 3
public int getAge()
{
   return age;
}

// method 4
public String getColor()
{
   return color;
}

@Override
public String toString()
{
   return("Hi my name is "+ this.getName()+
         ".\nMy breed,age and color are " +
         this.getBreed()+"," + this.getAge()+
         ","+ this.getColor());
}
```

```
    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}
```

Output:

Hi my name is tuffy.
My breed,age and color are papillon,5,white
This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler differentiates the constructors based on the number and the type of the arguments. The constructor in the Dog class takes four arguments. The following statement provides "tuffy","papillon",5,"white" as values for those arguments:

```
Dog tuffy = new Dog("tuffy","papillon",5, "white");
```

The result of executing this statement can be illustrated as :
Untitled

Note : All classes have at least one constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent's no-argument constructor (as it contain only one statement i.e super();), or the Object class constructor if the class has no other parent (as Object class is parent of all classes either directly or indirectly).

Ways to create object of a class

There are four ways to create objects in java.Strictly speaking there is only one way(by using new keyword),and the rest internally use new keyword.

Using new keyword : It is the most common and general way to create object in java. Example:

```
// creating object of class Test
Test t = new Test();
```

Using Class.forName(String className) method : There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name.We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

```
// creating object of public class Test
```

```
// consider class Test present in com.p1 package
Test obj = (Test)Class.forName("com.p1.Test").newInstance();
```
Using clone() method: clone() method is present in Object class. It creates and returns a copy of the object.

```
// creating object of class Test
Test t1 = new Test();

// creating clone of above object
Test t2 = (Test)t1.clone();
```

Deserialization : De-serialization is technique of reading an object from the saved state in a file. Refer Serialization/De-Serialization in java

```
FileInputStream file = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(file);
Object obj = in.readObject();
```
Creating multiple objects by one type only (A good practice)

In real-time, we need different objects of a class in different methods. Creating a number of references for storing them is not a good practice and therefore we declare a static reference variable and use it whenever required. In this case,wastage of memory is less. The objects that are not referenced anymore will be destroyed by Garbage Collector of java. Example:

```
Test test = new Test();
test = new Test();
```
In inheritance system, we use parent class reference variable to store a sub-class object. In this case, we can switch into different subclass objects using same referenced variable. Example:
```
class Animal {}

class Dog extends Animal {}
class Cat extends Animal {}

public class Test
{
   // using Dog object
   Animal obj = new Dog();

   // using Cat object
   obj = new Cat();
}
```
Anonymous objects

Anonymous objects are the objects that are instantiated but are not stored in a reference variable.

They are used for immediate method calling.
They will be destroyed after method calling.
They are widely used in different libraries. For example, in AWT libraries, they are used to perform some action on capturing an event(eg a key press).
In example below, when a key is button(referred by the btn) is pressed, we are simply creating anonymous object of EventHandler class for just calling handle method.

```
btn.setOnAction(new EventHandler()
{
   public void handle(ActionEvent event)
   {
      System.out.println("Hello World!");
   }
});
```

3. **How many types of inheritances does java supports with out interface. Explain with sample program?**

nheritance in Java
Inheritance
Types of Inheritance
Why multiple inheritance is not possible in Java in case of class?
Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Why use inheritance in java
For Method Overriding (so runtime polymorphism can be achieved).
For Code Reusability.
Terms used in Inheritance
Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example

Inheritance in Java

As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Test it Now

```
 Programmer salary is:40000.0
 Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

Types of inheritance in Java
Note: Multiple inheritance is not supported in Java through class.
When one class inherits multiple classes, it is known as multiple inheritance. For Example:

Multiple inheritance in Java
Single Inheritance Example
File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example
File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example
File: TestInheritance3.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Q) Why multiple inheritance is not supported in java?
To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
```

```
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

4. Describe three uses of final with example.

Final keyword in Java has three different uses: create constants, prevent inheritance and prevent methods from being overridden. Following is a list of uses of final keyword.

Using final to define constants: If you want to make a local variable, class variable (static field), or instance variable (non-static filed) constant, declare it final. A final variable may only be assigned to once and its value will not change and can help avoid programming errors.

Once a final variable has been assigned, it always contains the same value. If a final variable holds a reference to an object, then the state of the object may be changed by operations on the object, but the variable will always refer to the same object.

This also applies to arrays, because arrays are objects; if a final variable holds a reference to an array, then the components of the array may be changed by operations on the array, but the variable will always refer to the same array.

Using final to prevent inheritance: If you find a class's definition is complete and you don't want it to be sub-classed, declare it final. A final class cannot be inherited, therefore, it will be a compile-time error if the name of a final class appears in the extends clause of another class declaration; this implies that a final class cannot have any subclasses.

It is a compile-time error if a class is declared both final and abstract, because the implementation of such a class could never be completed.

Because a final class never has any subclasses, the methods of a final class are never overridden

Using final to prevent overriding: When a class is extended by other classes, its methods can be overridden for reuse. There may be circumstances when you want to prevent a particular method from being overridden, in that case, declare that method final. Methods declared as final cannot be overridden.

Using final for method arguments: Formal parameters of a method can be declared final to prevent them from accidental changes during the execution of method body.

5. What is virtual function. Can we call virtual function in constructor?

Calling virtual functions from constructors is problematic, and this problem can manifest itself in many ways. In this column, we'll take a look at this problem, with specific examples.
Last year, I bought a BlackBerry mobile. It came with software that can be installed on a PC, with which one can transfer songs, data, etc., from the PC to the mobile. When I installed the software and started it, it promptly crashed with the exception: "pure virtual function call"! Surprisingly, over a period of five years, I've faced the same problem many times, and some of the screenshots I've taken from different software are shown in Figures 1 to 3.
FireFoxPureVirtualCallError
Figure 1: FireFoxPureVirtualCallError

PureVirtCall
Figure 2: PureVirtCall

PureVirtCallAcrobat
Figure 3: PureVirtCallAcrobat

Note that this behaviour is not specific to Windows software; software compiled with GCC on Linux will fail with a similar exception. Now, let us dig deeper, to understand this software bug.
Virtual functions are resolved based on the runtime type of the object on which they are called. If we try invoking virtual methods from constructors or destructors, it is problematic. Why?

Consider the case of calling a virtual function from a constructor. When the base class constructor executes, the derived object is not constructed yet. If there is a virtual function call that is supposed to bind to the derived type, how can it be handled?

The ways in which OO languages handle this situation differ. In C++, the virtual function is treated as non-virtual, and the base type method is called. In Java, the call is resolved to the derived type. Both these approaches can cause unintuitive results. Let us first discuss the C++ case, and then move on to the Java case.

In C++, if you try the following program, it will print "Inside base::vfun" since the virtual function is resolved to the base type (i.e., static type itself, instead of the dynamic type):

```
struct base {
   base() {
      vfun();
   }
   virtual void vfun() {
```

```
        cout << "Inside base::vfunn";
    }
};
struct deri : base {
    virtual void vfun() {
        cout << "Inside deri::vfunn";
    }
};
int main(){
    deri d;
}
```

Now, how about this program:

```
struct base {
    base() {
        base * bptr = this;
        bptr->bar();
        // even simpler ...
        ((base*)(this))->bar();
    }
    virtual void bar() =0;
};
struct deri: base {
    void bar(){ }
};
int main() {
    deri d;
}
```

Now, you'll get the "pure virtual function call" exception thrown by the C++ runtime, which is similar to the three screenshots we saw earlier! In this case, the bar() method is a pure virtual function, which means that it is not defined yet (it is defined later, in a derived class). However, since we are invoking bar() from the base class constructor, it tries to call the pure virtual function; it is not possible to invoke a function that is not defined, and hence it results in a runtime exception (technically, it is "undefined behaviour").

Note how we invoked bar() in the base class constructor — it is after casting the this pointer into the (base *) type. If we attempt to directly call a pure virtual function, the compiler will give a compile-time error.

Now, let's look at a simple Java example. Can you predict its output?

```
class Base {
    public Base() {
        foo();
    }
    public void foo() {
```

```
        System.out.println("In Base's foo ");
    }
}
class Derived extends Base {
    public Derived() {
        i = new Integer(10);
    }
    public void foo() {
        System.out.println("In Derived's foo " + i.toString() );
    }
    private Integer i;
}
class Test {
    public static void main(String [] s) {
        new Derived().foo();
    }
}
```

The program crashes with a NullPointerException! Why? As I mentioned earlier, in Java, virtual functions are resolved to the dynamic type. Here, foo is a virtual function (in Java, all non-static, non-final methods are virtual) and we try invoking it from the constructor. Since it resolves to the dynamic type, the derived version of foo is called.

Remember that we are still executing the base class constructor, and the derived constructor is yet to execute. Hence the private variable i inside Derived is not initialised yet (and all reference type variables are initialised to null in Java). Hence, the call i.toString() results in accessing the yet-to-be-initialised Derived object, and results in a NullPointerException.

Calling virtual functions from constructors/destructors is risky, no matter which OO language we use. Even if the program works in cases where virtual functions are called from constructors, the program can suddenly start failing if we extend the base classes in which such calls are present. Hence, it is a bad programming practice to call virtual functions from constructors/destructors, and most static analysers warn about this problem.

6. Explain the life cycle of thread.

Life cycle of a Thread (Thread States)
Life cycle of a thread
New
Runnable
Running
Non-Runnable (Blocked)
Terminated
A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

New
Runnable
Running
Non-Runnable (Blocked)
Terminated
Java thread life cycle

1) New
The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Runnable
The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running
The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)
This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated
A thread is in terminated or dead state when its run() method exits.

7 What is Applet? How it is different from application?

All Java programs are classified as Applications and Applets. While applications are stand-alone Java programs that run directly on your machine, applets are specific programs that require use of a browser and should be included in an HTML web document.

In simple terms, application programs run with the help of a virtual machine independent of any security restrictions, whereas an applet cannot run without the help of a browser and is subjected to more harsh security restrictions in terms of network access. You can say, applets are kind of an internet application that doesn't require any kind of deployment procedure or installation. Let's study the difference between the two in detail.

Difference between Application and Applet

What is an Application?
It is a stand-alone Java program that runs with the support of a virtual machine in a client or server side. Also referred to as an application program, a Java application is designed to perform a specific function to run on any Java-compatible virtual machine regardless of the computer

architecture. An application is either executed for the user or for some other application program. Examples of Java applications include database programs, development tools, word processors, text and image editing programs, spreadsheets, web browsers etc.

Java applications can run with or without graphical user interface (GUI). It's a broad term used to define any kind of program in Java, but limited to the programs installed on your machine. Any application program can access any data or information or any resources available on the system without any security restrictions. Java application programs run by starting the Java interpreter from the command prompt and are compiled using the javac command and run using the java command. Every application program generally stays on the machine on which they are deployed. It has a single start point which has a main() method.

Difference between Application and Applet-1

What is an Applet?
Unlike a Java application program, an applet is specifically designed to be executed within an HTML web document using an external API. They are basically small programs – more like the web version of an application – that require a Java plugin to run on client browser. They run on the client side and are generally used for internet computing. You can execute a Java applet in a HTML page exactly as you would include an image in a web page. When you see a HTML page with an applet in a Java-enabled web browser, the applet code gets transferred to the system and is finally run by the Java-enabled virtual machine on the browser.

Applets are also compiled using the javac command but can only run using the appletviewer command or with a browser. A Java applet is capable of performing all kinds of operations such as play sounds, display graphics, perform arithmetic operations, create animated graphics, etc. You can integrate an applet into a web page either locally or remotely. You can either create your own applets locally or develop them externally. When stored on a local system, it's called a local applet. The ones which are stored on a remote location and are developed externally are called remote applets.

Browsers come with Java Runtime environment (JRE) to execute applets and these browsers are called Java-enabled browsers. The web page contains tags which specify the name of the applet and its URL (Uniform Resource Locator) – the unique location where the applet bytecodes reside on the World Wide Web. In simple terms, URLs refer to the files on some machine or network. Unlike applications, Java applets are executed in a more restricted environment with harsh security restrictions. They cannot access the resources on the system except the browser-specific services.

Difference between Application and Applet
Definition of Application and Applet – Applets are feature rich application programs that are specifically designed to be executed within an HTML web document to execute small tasks or just part of it. Java applications, on the other hand, are stand-alone programs that are designed to run on a stand-alone machine without having to use a browser.
Execution of Application and Applet– Applications require main method() to execute the code from the command line, whereas an applet does not require main method() for execution. An

applet requires an HTML file before its execution. The browser, in fact, requires a Java plugin to run an applet.
Compilation of Application and Applet–Application programs are compiled using the "javac" command and further executed using the java command. Applet programs, on the other hand, are also compiled using the "javac" command but are executed either by using the "appletviewer" command or using the web browser.
Security Access of Application and Applet – Java application programs can access all the resources of the system including data and information on that system, whereas applets cannot access or modify any resources on the system except only the browser specific services.
Restrictions of Application and Applet – Unlike applications, applet programs cannot be run independently, thus require highest level of security. However, they do not require any specific deployment procedure during execution. Java applications, on the other hand, run independently and do not require any security as they are trusted.
Application vs. Applet : Comparison Table
Application Applet
Applications are stand-alone programs that can be run independently without having to use a web browser. Applets are small Java programs that are designed to be included in a HTML web document. They require a Java-enabled browser for execution.
Java applications have full access to local file system and network. Applets have no disk and network access.
It requires a main method() for its execution.It does not require a main method() for its execution.
Applications can run programs from the local system. Applets cannot run programs from the local machine.
An application program is used to perform some task directly for the user. An applet program is used to perform small tasks or part of it.
It can access all kinds of resources available on the system.

8 What is synchronization? When do we use it? Explain by giving the different ways to implement synchronization?

h Exception Handling
Chained Exceptions
Multithreading in Java
Multithreading
Thread class
Creating a thread
Joining a thread
Synchronization
Interthread Communication
Advanced topics
Enumerations
Autoboxing and Unboxing
Java I/O Stream
Serialization
Java Networking

Generics

Collection Framework
Java GUI
Reflection API
RMI Application
JDBC
Synchronization
At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax :

```
synchronized (object)
{
//statement to be synchronized
}
```
Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Why we use Syncronization ?
If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads T1 and T2, T1 starts execution and save certain values in a file temporary.txt which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using temporary.txt file, this file will be locked(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods
Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Example with no Synchronization

```
class First
{
 public void display(String msg)
 {
```

```
  System.out.print ("["+msg);
  try
  {
   Thread.sleep(1000);
  }
  catch(InterruptedException e)
  {
   e.printStackTrace();
  }
  System.out.println ("]");
 }
}

class Second extends Thread
{
 String msg;
 First fobj;
 Second (First fp,String str)
 {
  fobj = fp;
  msg = str;
  start();
 }
 public void run()
 {
  fobj.display(msg);
 }
}

public class Syncro
{
 public static void main (String[] args)
 {
  First fnew = new First();
  Second ss = new Second(fnew, "welcome");
  Second ss1= new Second (fnew,"new");
  Second ss2 = new Second(fnew, "programmer");
 }
}

[welcome [ new [ programmer]
]
]
```

In the above program, object fnew of class First is shared by all the three running threads(ss, ss1 and ss2) to call the shared method(void display). Hence the result is unsynchronized and such situation is called Race condition.

Synchronized Keyword

To synchronize above program, we must synchronize access to the shared display() method, making it available to only one thread at a time. This is done by using keyword synchronized with display() method.

synchronized void display (String msg)

Using Synchronized block

If you have to synchronize access to an object of a class or you only want a part of a method to be synchronized to an object then you can use synchronized block for it.

```
class First
{
 public void display(String msg)
 {
  System.out.print ("["+msg);
  try
  {
   Thread.sleep(1000);
  }
  catch(InterruptedException e)
  {
   e.printStackTrace();
  }
  System.out.println ("]");
 }
}

class Second extends Thread
{
 String msg;
 First fobj;
 Second (First fp,String str)
 {
  fobj = fp;
  msg = str;
  start();
 }
 public void run()
 {
  synchronized(fobj)       //Synchronized block
  {
```

```
   fobj.display(msg);
  }
 }
}

public class Syncro
{
 public static void main (String[] args)
 {
  First fnew = new First();
  Second ss = new Second(fnew, "welcome");
  Second ss1= new Second (fnew,"new");
  Second ss2 = new Second(fnew, "programmer");
 }
}
```

[welcome]
[new]
[programmer]

Because of synchronized block this program gives the expected output.

Difference between synchronized keyword and synchronized block
When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

Which is more preferred - Synchronized method or Synchronized block?
In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

9 How to create Font object?

Important text can be made eye-catching by displaying in attractive fonts. A font represents a combination of a font name (like Monospaced), font style (like bold) and font size (like 20).

Following is the class signature

public class Font extends Object implements Serializable
The Font constructor takes three parameters to create a Font object.

Font f1 = new Font(String fontName, int fontStyle, int fontSize);
To give the font style as an integer value, the Font class comes with 3 symbolic constants.

public final static int PLAIN = 0;
public final static int BOLD = 1;
public final static int ITALIC = 2;

There are five font style attributes supported by JVM are

1. Monospaced
2. Serif
3. SansSerif
4. Dialog
5. DialogInput

If anyone other than the above five mentioned in font object creation, it is not an error, but gives the default Dialog, plain and size 12.

Java AWT Create Font Object – 4 styles
Font f1 = new Font("Monospaced", Font.BOLD, 15);
Font f2 = new Font("Serif", Font.ITALIC, 20);
Font f3 = new Font("SansSerif", Font.PLAIN, 19);
Font f4 = new Font("DialogInput", Font.BOLD + Font.ITALIC, 25);

The following example shows how to create Font object and apply to the string.

```
import java.awt.*;
public class UsingFonts extends Frame
{
 public void paint(Graphics g)
 {
  Font f1 = new Font("Monospaced", Font.BOLD, 12);
  g.setFont(f1);

  g.drawString("Font Particulars: " + g.getFont(), 15, 60);
  g.drawString("Font Name: " + f1.getFontName(), 15, 80);
  g.drawString("Font Style: " + f1.getStyle(), 15, 100);
  g.drawString("Font Size: " + f1.getSize(), 15, 120);
  g.drawString("isBold(): " + f1.isBold(), 15, 140);
  g.drawString("isItalic(): " + f1.isItalic(), 15, 160);
  g.drawString("isPlain(): " + f1.isItalic(), 15, 180);
 }
 public static void main(String args[])
 {
```

```
    Frame myFrame = new UsingFonts();
    myFrame.setSize(600, 250);
    myFrame.setVisible(true);
  }
}
import java.awt.*;
public class UsingFonts extends Frame
{
  public void paint(Graphics g)
  {
    Font f1 = new Font("Monospaced", Font.BOLD, 12);
    g.setFont(f1);

    g.drawString("Font Particulars: " + g.getFont(), 15, 60);
    g.drawString("Font Name: " + f1.getFontName(), 15, 80);
    g.drawString("Font Style: " + f1.getStyle(), 15, 100);
    g.drawString("Font Size: " + f1.getSize(), 15, 120);
    g.drawString("isBold(): " + f1.isBold(), 15, 140);
    g.drawString("isItalic(): " + f1.isItalic(), 15, 160);
    g.drawString("isPlain(): " + f1.isItalic(), 15, 180);
  }
  public static void main(String args[])
  {
    Frame myFrame = new UsingFonts();
    myFrame.setSize(600, 250);
    myFrame.setVisible(true);
  }
}
```

Java AWT Create Font Object

```
Font f1 = new Font("Monospaced", Font.BOLD, 12);
g.setFont(f1);
```

A Font object f1 is created with font name of Monospaced, style bold and size 12. setFont() method of Graphics class sets the font to the text. For a similar program with many font and color objects refer ColorsAndFonts.java.

Note: Observe another style of creating frame. Frame is created in the main() method instead of conventional constructor as done in other programs.

The frame you get do not close when clicked over the close icon on the title bar of the frame. It requires extra code close icon to work.

10 Write short notes on Graphics and Font class?

Java AWT Graphics Example
Introduction
The Java 2D API is powerful and complex. However, the vast majority of uses for the Java 2D API utilize a small subset of its capabilities encapsulated in the java.awt.Graphics class. This lesson covers the most common needs of applications developers.
Most methods of the Graphics class can be divided into two basic groups:

Draw and fill methods, enabling you to render basic shapes, text, and images.
Attributes setting methods, which affect how that drawing and filling appears.
Methods such as setFont and setColor define how draw and fill methods render.
Drawing methods include:

drawString – For drawing text
drawImage – For drawing images
drawLine, drawArc, drawRect, drawOval, drawPolygon – For drawing geometric shapes
Depending on your current need, you can choose one of several methods in the Graphics class based on the following criteria:

Whether you want to render the image at the specified location in its original size or scale it to fit inside the given rectangle.
Whether you prefer to fill the transparent areas of the image with color or keep them transparent.
Fill methods apply to geometric shapes and include fillArc, fillRect, fillOval, fillPolygon. Whether you draw a line of text or an image, remember that in 2D graphics every point is determined by its x and y coordinates. All of the draw and fill methods need this information which determines where the text or image should be rendered..

For example, to draw a line, an application calls the following:

AWTGraphicsExample.java
java.awt.Graphics.drawLine(int x1, int y1, int x2, int y2)
2. The java.awt.Graphics Class: Graphics Context and Custom Painting
A graphics context provides the capabilities of drawing on the screen. The graphics context maintains states such as the color and font used in drawing, as well as interacting with the underlying operating system to perform the drawing. In Java, custom painting is done via the java.awt.Graphics class, which manages a graphics context, and provides a set of device-independent methods for drawing texts, figures and images on the screen on different platforms.

The java.awt.Graphics is an abstract class, as the actual act of drawing is system-dependent and device-dependent. Each operating platform will provide a subclass of Graphics to perform the actual drawing under the platform, but conform to the specification defined in Graphics.

2.1 Graphics Class' Drawing Methods
The Graphics class provides methods for drawing three types of graphical objects:

1.Text strings: via the drawString() method. Take note that System.out.println() prints to the system console, not to the graphics screen.
2.Vector-graphic primitives and shapes: via methods rawXxx() and fillXxx(), where Xxx could be Line, Rect, Oval, Arc, PolyLine, RoundRect, or 3DRect.
3.Bitmap images: via the drawImage() method.

AWTGraphicsExample.java
```
// Drawing (or printing) texts on the graphics screen:
drawString(String str, int xBaselineLeft, int yBaselineLeft);

// Drawing lines:
drawLine(int x1, int y1, int x2, int y2);
drawPolyline(int[] xPoints, int[] yPoints, int numPoint);

// Drawing primitive shapes:
drawRect(int xTopLeft, int yTopLeft, int width, int height);
drawOval(int xTopLeft, int yTopLeft, int width, int height);
drawArc(int xTopLeft, int yTopLeft, int width, int height, int startAngle, int arcAngle);
draw3DRect(int xTopLeft, int, yTopLeft, int width, int height, boolean raised);
drawRoundRect(int xTopLeft, int yTopLeft, int width, int height, int arcWidth, int arcHeight)
drawPolygon(int[] xPoints, int[] yPoints, int numPoint);

// Filling primitive shapes:
fillRect(int xTopLeft, int yTopLeft, int width, int height);
fillOval(int xTopLeft, int yTopLeft, int width, int height);
fillArc(int xTopLeft, int yTopLeft, int width, int height, int startAngle, int arcAngle);
fill3DRect(int xTopLeft, int, yTopLeft, int width, int height, boolean raised);
fillRoundRect(int xTopLeft, int yTopLeft, int width, int height, int arcWidth, int arcHeight)
fillPolygon(int[] xPoints, int[] yPoints, int numPoint);

// Drawing (or Displaying) images:
drawImage(Image img, int xTopLeft, int yTopLeft, ImageObserver obs);  // draw image with its size
drawImage(Image img, int xTopLeft, int yTopLeft, int width, int height, ImageObserver o);  // resize image on screen
```

2.2 Graphics Class' Methods for Maintaining the Graphics Context

The graphic context maintains states (or attributes) such as the current painting color, the current font for drawing text strings, and the current painting rectangular area (called clip). You can use the methods getColor(), setColor(), getFont(), setFont(), getClipBounds(), setClip() to get or set the color, font, and clip area. Any painting outside the clip area is ignored.

AWTGraphicsExample.java
```
// Graphics context's current color.
void setColor(Color c)
Color getColor()
```

// Graphics context's current font.
void setFont(Font f)
Font getFont()

// Set/Get the current clip area. Clip area shall be rectangular and no rendering is performed outside the clip area.
void setClip(int xTopLeft, int yTopLeft, int width, int height)
void setClip(Shape rect)
public abstract void clipRect(int x, int y, int width, int height) // intersects the current clip with the given rectangle
Rectangle getClipBounds() // returns an Rectangle
Shape getClip() // returns an object (typically Rectangle) implements Shape

2.3 Graphics Class' Other Methods

AWTGraphicsExample.java

void clearRect(int x, int y, int width, int height)
  // Clear the rectangular area to background
void copyArea(int x, int y, int width, int height, int dx, int dy)
  // Copy the rectangular area to offset (dx, dy).
void translate(int x, int y)
  // Translate the origin of the graphics context to (x, y). Subsequent drawing uses the new origin.
FontMetrics getFontMetrics()
FontMetrics getFontMetrics(Font f)
  // Get the FontMetrics of the current font / the specified font

2.4 Graphics Coordinate System

In Java Windowing Subsystem (like most of the 2D Graphics systems), the origin (0,0) is located at the top-left corner.

EACH component/container has its own coordinate system, ranging for (0,0) to (width-1, height-1) as illustrated.

You can use method getWidth() and getHeight() to retrieve the width and height of a component/container. You can use getX() or getY() to get the top-left corner (x,y) of this component's origin relative to its parent.

3 Custom Painting Template

Under Swing, custom painting is usually performed by extending (i.e., subclassing) a JPanel as the drawing canvas and override the paintComponent(Graphics g) method to perform your own drawing with the drawing methods provided by the Graphics class. The Java Windowing Subsystem invokes (calls back) paintComponent(g) to render the JPanel by providing the current graphics context g, which can be used to invoke the drawing methods.

The extended JPanel is often programmed as an inner class of a JFrame application to facilitate access of private variables/methods. Although we typically draw on the JPanel, you can in fact draw on any JComponent (such as JLabel, JButton).
The custom painting code template is as follows:

AWTGraphicsExample.java

```
import java.awt.*;       // Using AWT's Graphics and Color
import java.awt.event.*; // Using AWT event classes and listener interfaces
import javax.swing.*;    // Using Swing's components and containers

/** Custom Drawing Code Template */
// A Swing application extends javax.swing.JFrame
public class CGTemplate extends JFrame {
   // Define constants
   public static final int CANVAS_WIDTH  = 640;
   public static final int CANVAS_HEIGHT = 480;

   // Declare an instance of the drawing canvas,
   // which is an inner class called DrawCanvas extending javax.swing.JPanel.
   private DrawCanvas canvas;

   // Constructor to set up the GUI components and event handlers
   public CGTemplate() {
      canvas = new DrawCanvas();    // Construct the drawing canvas
      canvas.setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));

      // Set the Drawing JPanel as the JFrame's content-pane
      Container cp = getContentPane();
      cp.add(canvas);
      // or "setContentPane(canvas);"

      setDefaultCloseOperation(EXIT_ON_CLOSE);   // Handle the CLOSE button
      pack();              // Either pack() the components; or setSize()
      setTitle(".....");  // "super" JFrame sets the title
      setVisible(true);    // "super" JFrame show
   }

   /**
    * Define inner class DrawCanvas, which is a JPanel used for custom drawing.
    */
   private class DrawCanvas extends JPanel {
      // Override paintComponent to perform your own painting
      @Override
      public void paintComponent(Graphics g) {
         super.paintComponent(g);     // paint parent's background
         setBackground(Color.BLACK);  // set background color for this JPanel

         // Your custom painting codes. For example,
         // Drawing primitive shapes
         g.setColor(Color.YELLOW);    // set the drawing color
         g.drawLine(30, 40, 100, 200);
```

```
        g.drawOval(150, 180, 10, 10);
        g.drawRect(200, 210, 20, 30);
        g.setColor(Color.RED);       // change the drawing color
        g.fillOval(300, 310, 30, 50);
        g.fillRect(400, 350, 60, 50);
        // Printing texts
        g.setColor(Color.WHITE);
        g.setFont(new Font("Monospaced", Font.PLAIN, 12));
        g.drawString("Testing custom drawing ...", 10, 20);
      }
    }

    // The entry main method
    public static void main(String[] args) {
      // Run the GUI codes on the Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
          new CGTemplate(); // Let the constructor do the job
        }
      });
    }
}
```

Dissecting the Program

Custom painting is performed by extending a JPanel (called DrawCanvas) and overrides the paintComponent(Graphics g) method to do your own drawing with the drawing methods provided by the Graphics class.

DrawCanvas is designed as an inner class of this JFrame application, so as to facilitate access of the private variables/methods.

Java Windowing Subsystem invokes (calls back) paintComponent(g) to render the JPanel, with the current graphics context in g, whenever there is a need to refresh the display (e.g., during the initial launch, restore, resize, etc). You can use the drawing methods (g.drawXxx() and g.fillXxx()) on the current graphics context g to perform custom painting on the JPanel.

The size of the JPanel is set via the setPreferredSize(). The JFrame does not set its size, but packs the components contained via pack().

In the main(), the constructor is called in the event-dispatch thread via static method javax.swing.SwingUtilities.invokeLater() (instead of running in the main thread), to ensure thread-safety and avoid deadlock, as recommended by the Swing developers.

3.1 Refreshing the Display via repaint()

At times, we need to explicitly refresh the display (e.g., in game and animation). We shall NOT invoke paintComponent(Graphics) directly. Instead, we invoke the JComponent's repaint() method. The Windowing Subsystem will in turn call back the paintComponent() with the current Graphics context and execute it in the event-dispatching thread for thread safety. You can repaint() a particular JComponent (such as a JPanel) or the entire JFrame. The children contained within the JComponent will also be repainted.

4. Colors and Fonts

4.1 java.awt.Color

The class java.awt.Color provides 13 standard colors as named-constants. They are: Color.RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, and PINK. (In JDK 1.1, these constant names are in lowercase, e.g., red. This violates the Java naming convention for constants. In JDK 1.2, the uppercase names are added. The lowercase names were not removed for backward compatibility.)

You can use the toString() to print the RGB values of these color (e.g., System.out.println(Color.RED)):

AWTGraphicsExample.java

```
RED       : java.awt.Color[r=255, g=0,   b=0]
GREEN     : java.awt.Color[r=0,   g=255, b=0]
BLUE      : java.awt.Color[r=0,   g=0,   b=255]
YELLOW    : java.awt.Color[r=255, g=255, b=0]
MAGENTA   : java.awt.Color[r=255, g=0,   b=255]
CYAN      : java.awt.Color[r=0,   g=255, b=255]
WHITE     : java.awt.Color[r=255, g=255, b=255]
BLACK     : java.awt.Color[r=0,   g=0,   b=0]
GRAY      : java.awt.Color[r=128, g=128, b=128]
LIGHT_GRAY: java.awt.Color[r=192, g=192, b=192]
DARK_GRAY : java.awt.Color[r=64,  g=64,  b=64]
PINK      : java.awt.Color[r=255, g=175, b=175]
ORANGE    : java.awt.Color[r=255, g=200, b=0]
```

To retrieve the individual components, you can use getRed(), getGreen(), getBlue(), getAlpha(), etc.

To set the background and foreground (text) color of a component/container, you can invoke:

AWTGraphicsExample.java

```
JLabel label = new JLabel("Test");
label.setBackground(Color.LIGHT_GRAY);
label.setForeground(Color.RED);
```

4.2 java.awt.Font

The class java.awt.Font represents a specific font face, which can be used for rendering texts. You can use the following constructor to construct a Font instance:

AWTGraphicsExample.java

```
public Font(String name, int style, int size);
// name:  Family name "Dialog", "DialogInput", "Monospaced", "Serif", or "SansSerif" or
//        Physical font found in this GraphicsEnvironment.
//                You can also use String constants Font.DIALOG, Font.DIALOG_INPUT,
Font.MONOSPACED,
```

```
//          Font.SERIF, Font.SANS_SERIF (JDK 1.6)
// style: Font.PLAIN, Font.BOLD, Font.ITALIC or Font.BOLD|Font.ITALIC (Bit-OR)
// size:  the point size of the font (in pt) (1 inch has 72 pt).
```

You can use the setFont() method to set the current font for the Graphics context g for rendering texts. For example,

AWTGraphicsExample.java

```
Font myFont1 = new Font(Font.MONOSPACED, Font.PLAIN, 12);
Font myFont2 = new Font(Font.SERIF, Font.BOLD | Font.ITALIC, 16);  // bold and italics
JButton btn = new JButton("RESET");
btn.setFont(myFont1);
JLabel lbl = new JLabel("Hello");
lbl.setFont(myFont2);
.....
g.drawString("In default Font", 10, 20);     // in default font
Font myFont3 = new Font(Font.SANS_SERIF, Font.ITALIC, 12);
g.setFont(myFont3);
g.drawString("Using the font set", 10, 50);  // in myFont3
```

Font's Family Name vs. Font Name

A font could have many faces (or style), e.g., plain, bold or italic. All these faces have similar typographic design. The font face name, or font name for short, is the name of a particular font face, like "Arial", "Arial Bold", "Arial Italic", "Arial Bold Italic". The font family name is the name of the font family that determines the typographic design across several faces, like "Arial". For example,

AWTGraphicsExample.java

```
java.awt.Font[family=Arial,name=Arial,style=plain,size=1]
java.awt.Font[family=Arial,name=Arial Bold,style=plain,size=1]
java.awt.Font[family=Arial,name=Arial Bold Italic,style=plain,size=1]
java.awt.Font[family=Arial,name=Arial Italic,style=plain,size=1]
```

Logical Font vs. Physical Font

JDK supports these logical font family names: "Dialog", "DialogInput", "Monospaced", "Serif", or "SansSerif". JDK 1.6 provides these String constants: Font.DIALOG, Font.DIALOG_INPUT, Font.MONOSPACED, Font.SERIF, Font.SANS_SERIF.

Physical font names are actual font libraries such as "Arial", "Times New Roman" in the system.

GraphicsEnvironment's getAvailableFontFamilyNames() and getAllFonts()

You can use GraphicsEnvironment's getAvailableFontFamilyNames() to list all the font family names; and getAllFonts() to construct all Font instances (with font size of 1 pt). For example,

```
GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
```

AWTGraphicsExample.java

```
// Get all font family name in a String[]
String[] fontNames = env.getAvailableFontFamilyNames();
for (String fontName : fontNames) {
   System.out.println(fontName);
}

// Construct all Font instance (with font size of 1)
Font[] fonts = env.getAllFonts();
for (Font font : fonts) {
   System.out.println(font);
}
```

11. What is Layout manager? Explain various layout managers supported by swing?

Several AWT and Swing classes provide layout managers for general use:

BorderLayout
BoxLayout
CardLayout
FlowLayout
GridBagLayout
GridLayout
GroupLayout
SpringLayout
This section shows example GUIs that use these layout managers, and tells you where to find the how-to page for each layout manager. You can find links for running the examples in the how-to pages and in the example index.

Note: This lesson covers writing layout code by hand, which can be challenging. If you are not interested in learning all the details of layout management, you might prefer to use the GroupLayout layout manager combined with a builder tool to lay out your GUI. One such builder tool is the NetBeans IDE. Otherwise, if you want to code by hand and do not want to use GroupLayout, then GridBagLayout is recommended as the next most flexible and powerful layout manager.
If you are interested in using JavaFX to create your GUI, see Working With Layouts in JavaFX.

BorderLayout
A picture of a GUI that uses BorderLayout
Every content pane is initialized to use a BorderLayout. (As Using Top-Level Containers explains, the content pane is the main container in all frames, applets, and dialogs.) A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions. For further details, see How to Use BorderLayout.

BoxLayout
A picture of a GUI that uses BoxLayout
The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components. For further details, see How to Use BoxLayout.

CardLayout
A picture of a GUI that uses CardLayout Another picture of the same layout

The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays. An alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI. For further details, see How to Use CardLayout.

FlowLayout
A picture of a GUI that uses FlowLayout
FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide. Both panels in CardLayoutDemo, shown previously, use FlowLayout. For further details, see How to Use FlowLayout.

GridBagLayout
A picture of a GUI that uses GridBagLayout
GridBagLayout is a sophisticated, flexible layout manager. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. The rows in the grid can have different heights, and grid columns can have different widths. For further details, see How to Use GridBagLayout.

GridLayout
A picture of a GUI that uses GridLayout
GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns. For further details, see How to Use GridLayout.

GroupLayout
A picture of a GUI that uses GroupLayout
GroupLayout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. Consequently, however, each component needs to be defined twice in the layout. The Find window shown above is an example of a GroupLayout. For further details, see How to Use GroupLayout.

SpringLayout
A picture of a GUI that uses SpringLayout
Another GUI that uses SpringLayout

SpringLayout is a flexible layout manager designed for use by GUI builders. It lets you specify precise relationships between the edges of components under its control. For example, you might define that the left edge of one component is a certain distance (which can be dynamically calculated) from the right edge of a second component. SpringLayout lays out the children of its associated container according to a set of constraints, as shall be seen in How to Use SpringLayout.

12 Explain different types of dialog boxes that can be crated by JOptionPane class?

JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something. For information about using JOptionPane, see How to Make Dialogs, a section in The Java Tutorial.
While the JOptionPane class may appear complex because of the large number of methods, almost all uses of this class are one-line calls to one of the static showXxxDialog methods shown below:

Method Name Description
showConfirmDialog Asks a confirming question, like yes/no/cancel.
showInputDialog Prompt for some input.
showMessageDialog Tell the user about something that has happened.
showOptionDialog The Grand Unification of the above three.
Each of these methods also comes in a showInternalXXX flavor, which uses an internal frame to hold the dialog box (see JInternalFrame). Multiple convenience methods have also been defined -- overloaded versions of the basic methods that use different parameter lists.
All dialogs are modal. Each showXxxDialog method blocks the caller until the user's interaction is complete.

icon message
input value
option buttons The basic appearance of one of these dialog boxes is generally similar to the picture at the right, although the various look-and-feels are ultimately responsible for the final result. In particular, the look-and-feels will adjust the layout to accommodate the option pane's ComponentOrientation property.
Parameters:
The parameters to these methods follow consistent patterns:

parentComponent
Defines the Component that is to be the parent of this dialog box. It is used in two ways: the Frame that contains it is used as the Frame parent for the dialog box, and its screen coordinates are used in the placement of the dialog box. In general, the dialog box is placed just below the component. This parameter may be null, in which case a default Frame is used as the parent, and the dialog will be centered on the screen (depending on the L&F).
message
A descriptive message to be placed in the dialog box. In the most common usage, message is just a String or String constant. However, the type of this parameter is actually Object. Its interpretation depends on its type:

Object[]
An array of objects is interpreted as a series of messages (one per object) arranged in a vertical stack. The interpretation is recursive -- each object in the array is interpreted according to its type.
Component
The Component is displayed in the dialog.
Icon
The Icon is wrapped in a JLabel and displayed in the dialog.
others
The object is converted to a String by calling its toString method. The result is wrapped in a JLabel and displayed.
messageType
Defines the style of the message. The Look and Feel manager may lay out the dialog differently depending on this value, and will often provide a default icon. The possible values are:
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
optionType
Defines the set of option buttons that appear at the bottom of the dialog box:
DEFAULT_OPTION
YES_NO_OPTION
YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION
You aren't limited to this set of option buttons. You can provide any buttons you want using the options parameter.
options
A more detailed description of the set of option buttons that will appear at the bottom of the dialog box. The usual value for the options parameter is an array of Strings. But the parameter type is an array of Objects. A button is created for each object depending on its type:
Component
The component is added to the button row directly.
Icon
A JButton is created with this as its label.
other
The Object is converted to a string using its toString method and the result is used to label a JButton.
icon
A decorative icon to be placed in the dialog box. A default value for this is determined by the messageType parameter.
title
The title for the dialog box.
initialValue
The default selection (input value).
When the selection is changed, setValue is invoked, which generates a PropertyChangeEvent.

If a JOptionPane has configured to all input setWantsInput the bound property JOptionPane.INPUT_VALUE_PROPERTY can also be listened to, to determine when the user has input or selected a value.

When one of the showXxxDialog methods returns an integer, the possible values are:

YES_OPTION
NO_OPTION
CANCEL_OPTION
OK_OPTION
CLOSED_OPTION
Examples:
Show an error dialog that displays the message, 'alert':
JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);
Show an internal information dialog with the message, 'information':
JOptionPane.showInternalMessageDialog(frame, "information",
"information", JOptionPane.INFORMATION_MESSAGE);
Show an information panel with the options yes/no and message 'choose one':
JOptionPane.showConfirmDialog(null,
"choose one", "choose one", JOptionPane.YES_NO_OPTION);
Show an internal information dialog with the options yes/no/cancel and message 'please choose one' and title information:
JOptionPane.showInternalConfirmDialog(frame,
"please choose one", "information",
JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE);
Show a warning dialog with the options OK, CANCEL, title 'Warning', and message 'Click OK to continue':
Object[] options = { "OK", "CANCEL" };
JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning",
JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
null, options, options[0]);
Show a dialog asking the user to type in a String:
String inputValue = JOptionPane.showInputDialog("Please input a value");
Show a dialog asking the user to select a String:
Object[] possibleValues = { "First", "Second", "Third" };
Object selectedValue = JOptionPane.showInputDialog(null,
"Choose one", "Input",
JOptionPane.INFORMATION_MESSAGE, null,
possibleValues, possibleValues[0]);
Direct Use:
To create and use an JOptionPane directly, the standard pattern is roughly as follows:

```
JOptionPane pane = new JOptionPane(arguments);
pane.set.Xxxx(...); // Configure
JDialog dialog = pane.createDialog(parentComponent, title);
dialog.show();
```

```
Object selectedValue = pane.getValue();
if(selectedValue == null)
  return CLOSED_OPTION;
//If there is not an array of option buttons:
if(options == null) {
  if(selectedValue instanceof Integer)
     return ((Integer)selectedValue).intValue();
  return CLOSED_OPTION;
}
//If there is an array of option buttons:
for(int counter = 0, maxCounter = options.length;
   counter < maxCounter; counter++) {
   if(options[counter].equals(selectedValue))
   return counter;
}
return CLOSED_OPTION;
```

13 Explain MenuBar creation in Java?

To create menus, the java.awt package comes with mainly four classes – MenuBar, Menu, MenuItem and CheckboxMenuItem. All these four classes are not AWT components as they are not subclasses of java.awt.Component class. Infact, they are subclasses of java.awt.MenuComponent which is is no way connected in the hierarchy with Component class.

MenuBar: MenuBar holds the menus. MenuBar is added to frame with setMenuBar() method. Implicitly, the menu bar is added to the north (top) of the frame. MenuBar cannot be added to other sides like south and west etc.
Menu: Menu holds the menu items. Menu is added to frame with add() method. A sub-menu can be added to Menu.
MenuItem: MenuItem displays the actual option user can select. Menu items are added to menu with method addMenuItem(). A dull-colored line can be added in between menu items with addSeparator() method. The dull-colored line groups (or separates from other) menu items with similar functionality like cut, copy and paste.
CheckboxMenuItem: It differs from MenuItem in that it appears along with a checkbox. The selection can be done with checkbox selected.
Following is a simple Menus Java program with just two menus added with few menu items. User selection is displayed at DOS prompt for simplicity to get the concept. A big program we see later.

```
import java.awt.*;
import java.awt.event.*;

public class SimpleMenuExample extends Frame implements ActionListener
{
 Menu states, cities;
 public SimpleMenuExample()
 {
```

```
    MenuBar mb = new MenuBar();                 // begin with creating menu bar
    setMenuBar(mb);                             // add menu bar to frame

    states = new Menu("Indian States");             // create menus
    cities = new Menu("Indian Cities");

    mb.add(states);                             // add menus to menu bar
    mb.add(cities);

    states.addActionListener(this);             // link with ActionListener for event handling
    cities.addActionListener(this);

    states.add(new MenuItem("Himachal Pradesh"));
    states.add(new MenuItem("Rajasthan"));
    states.add(new MenuItem("West Bengal"));
    states.addSeparator();                      // separates from north Indian states from south
Indian
    states.add(new MenuItem("Andhra Pradesh"));
    states.add(new MenuItem("Tamilnadu"));
    states.add(new MenuItem("Karnataka"));

    cities.add(new MenuItem("Delhi"));
    cities.add(new MenuItem("Jaipur"));
    cities.add(new MenuItem("Kolkata"));
    cities.addSeparator();                      // separates north Indian cities from south Indian
    cities.add(new MenuItem("Hyderabad"));
    cities.add(new MenuItem("Chennai"));
    cities.add(new MenuItem("Bengaluru"));

    setTitle("Simple Menu Program");            // frame creation methods
    setSize(300, 300);
    setVisible(true);
  }
  public void actionPerformed(ActionEvent e)
  {
    String str = e.getActionCommand();              // know the menu item selected by the user
    System.out.println("You selected " + str);
  }
  public static void main(String args[])
  {
    new SimpleMenuExample();
  }
}
import java.awt.*;
import java.awt.event.*;
```

```
public class SimpleMenuExample extends Frame implements ActionListener
{
 Menu states, cities;
 public SimpleMenuExample()
 {
  MenuBar mb = new MenuBar();                    // begin with creating menu bar
  setMenuBar(mb);                                // add menu bar to frame

  states = new Menu("Indian States");            // create menus
  cities = new Menu("Indian Cities");

  mb.add(states);                                // add menus to menu bar
  mb.add(cities);

  states.addActionListener(this);                // link with ActionListener for event handling
  cities.addActionListener(this);

  states.add(new MenuItem("Himachal Pradesh"));
  states.add(new MenuItem("Rajasthan"));
  states.add(new MenuItem("West Bengal"));
  states.addSeparator();                         // separates from north Indian states from south
Indian
  states.add(new MenuItem("Andhra Pradesh"));
  states.add(new MenuItem("Tamilnadu"));
  states.add(new MenuItem("Karnataka"));

  cities.add(new MenuItem("Delhi"));
  cities.add(new MenuItem("Jaipur"));
  cities.add(new MenuItem("Kolkata"));
  cities.addSeparator();                         // separates north Indian cities from south Indian
  cities.add(new MenuItem("Hyderabad"));
  cities.add(new MenuItem("Chennai"));
  cities.add(new MenuItem("Bengaluru"));

  setTitle("Simple Menu Program");               // frame creation methods
  setSize(300, 300);
  setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
  String str = e.getActionCommand();             // know the menu item selected by the user
  System.out.println("You selected " + str);
 }
 public static void main(String args[])
 {
  new SimpleMenuExample();
```

```
  }
}
```

14 How can we add image to Applet?

Displaying Image in Applet
Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:
public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.
How to get the object of Image:
The java.applet.Applet class provides getImage() method that returns the object of Image.
Syntax:
public Image getImage(URL u, String image){}
Other required methods of Applet class to display image:
public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.
public URL getCodeBase(): is used to return the base URL.
Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;


public class DisplayImage extends Applet {

  Image picture;

  public void init() {
    picture = getImage(getDocumentBase(),"sonoo.jpg");
  }

  public void paint(Graphics g) {
    g.drawImage(picture, 30,30, this);
  }

  }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.
myapplet.html

```
<html>
<body>
```

```
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

download this example.

Next TopicAnimation-in-applet

<<prevnext>>

Learn Latest Tutorials
Computer Graphics Tutorial
C. Graphics

Automata Tutorial
Automata

Software Testing Tutorial
Testing

Numpy Tutorial
NumPy

Verbal Ability
Verbal A.

AWS Tutorial
AWS

Preparation
Aptitude
Aptitude

Logical Reasoning
Reasoning

Verbal Ability
Verbal A.

Interview Questions
Interview

B.Tech / MCA
DBMS tutorial
DBMS

Data Structures tutorial
DS

DAA tutorial
DAA

Operating System tutorial
OS

Computer Network tutorial
C. Network

Compiler Design tutorial
Compiler D.

Computer Organization and Architecture
COA

Discrete Mathematics Tutorial
D. Math.

html tutorial
Web Tech.

Cyber Security tutorial
Cyber Sec.

C Language tutorial
C

C++ tutorial
C++

Java tutorial
Java

.Net Framework tutorial
.Net

Python tutorial
Python

List of Programs
Programs

Control Systems tutorial
Control S.

15 Write Short Notes on URL, URLConnection?

next →← prev
Java URL
The Java URL class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

http://www.javatpoint.com/java-tutorial
A URL contains many information:

Protocol: In this case, http is the protocol.
Server name or IP Address: In this case, www.javatpoint.com is the server name.
Port Number: It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
File Name or directory name: In this case, index.jsp is the file name.
Commonly used methods of Java URL class
The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");

System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
```

```
System.out.println("File Name: "+url.getFile());

}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection class

```
1. import java.io.*;
2. import java.net.*;
3. public class URLConnectionExample {
4. public static void main(String[] args){
5. try{
6. URL url=new URL("http://www.javatpoint.com/java-tutorial");
7. URLConnection urlcon=url.openConnection();
8. InputStream stream=urlcon.getInputStream();
9. int i;
10. while((i=stream.read())!=-1){
11. System.out.print((char)i);
12. }
13. }catch(Exception e){System.out.println(e);}
```

14. }
15. }

16 What is the difference between TCP and UDP?

Difference between TCP and UDP in Java

What is the difference between TCP and UDP is a popular networking question from Java interviews? Though TCP or UDP is Java independent concept and very likely to be asked in other programming language interviews as well, many programmers not really understand them clearly. They sure have heard them because TCP and UDP are two of the most important transport protocol of internet, but when it comes to listing down the difference between them, they fail to mention key differences in terms of ordering, guaranteed delivery, speed, and usage. The biggest benefit of TCP/IP protocol is that it provides guaranteed to deliver of messages and in the order client sent them, that's very important when dealing with important messages e.g. order, trade, and booking messages. You cannot afford to lose them, neither you can process then out-of-sequence. The UDP protocol, on the other hand, provides the much-needed speed and can be used to implement a multicast network. In this article, I'll tell you the difference between TCP and UDP protocol from Java interview perspective.

Difference between TCP and UDP
Here are some important difference between TCP and UDP protocol from Java application developer's perspective.

1) TCP is a reliable stream oriented protocol as opposed to UDP which is not reliable and based upon datagram. This is actually the most significant difference between TCP and UDP as you cannot use the UDP for sending important messages which you can't afford to lose. Though there are some reliable protocols built over UDP e.g. TIBCO certified messaging which implement additional checks whether the message is delivered or not and then facilitate re-transmission.

2) One more important difference between TCP and UDP comes from speed. Since TCP is reliable and connection oriented it has lots of overhead as compared to UDP, which means TCP is slower than UDP and should not be used for transferring message where speed is critical e.g. live telecast, video or audio streaming. This is the reason UDP is popularly used in media transmission world.

3) Another structural difference between TCP and UDP is that data boundaries are preserved in case of UDP but not in the case of TCP because data is sent as it is as one message in case of UDP but TCP protocol can break and reassemble the data at sending and receiving end.

4) Another key difference between TCP and UDP protocol comes from the fact that TCP is a connection-oriented protocol but UDP is a connectionless protocol. What this mean is, before sending a message a connection is established between sender and receiver in TCP but no connection exists between the sender and receiver in UDP protocol.

TCP vs UDP differences

5) One more difference between UDP and TCP protocol which comes from how they work with ordering. TCP provides you order guarantee but UDP doesn't provide any ordering guarantee. For example, if Sender sends 3 messages than the receiver will receive those three messages in the same order, Sender, has sent, even if they are received at different order at receiver end TCP will ensure they are delivered to a client in the order they are sent by the sender. UDP doesn't provide this feature, which means it's possible for the last message to be received first and vice-versa.

6) TCP header size is larger than UDP header size due to excessive metadata information sent by TCP protocol. Those are required to ensure the guarantee provided by TCP protocol e.g. guaranteed ordered delivery. You can also read The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference 1st Edition, it's easy to read, interesting, and improve your fundamentals on TCP/IP and UDP protocol along with how to write reliable networking program in Java.

17 Write a program to implement multithreading?

```
class A extends Thread
{
	public void run()
	{
		System.out.println("A run called");
		for(int i=1;i<=100;i++)
		System.out.print(" A:"+i);
	}
}

class B extends Thread
{
	public void run()
	{
		System.out.println("B run called");
		for(int i=1;i<=100;i++)
				System.out.print(" B:"+i);
```

```
	}
}
class C extends Thread
{
	public void run()
	{
		System.out.println("C run called");
		for(int i=1;i<=100;i++)
				System.out.print(" C:"+i);
	}
}

class Thread3
{
	public static void main(String args[])
	{
		A obj1=new A();
		B obj2=new B();
		C obj3=new C();

		System.out.println("obj1 start called");
		obj1.start();
		System.out.println("obj2 start called");
		obj2.start();
		System.out.println("obj3 start called");
		obj3.start();
		System.out.println("main exiting");
	}
}
```

18 How do you set Priories for Threads

In a Multi threading environment, thread scheduler assigns processor to a thread based on priority of thread. Whenever we create a thread in Java, it always has some priority assigned to it. Priority can either be given by JVM while creating the thread or it can be given by programmer explicitly.
Accepted value of priority for a thread is in range of 1 to 10. There are 3 static variables defined in Thread class for priority.

public static int MIN_PRIORITY: This is minimum priority that a thread can have. Value for this is 1.
public static int NORM_PRIORITY: This is default priority of a thread if do not explicitly define it. Value for this is 5.
public static int MAX_PRIORITY: This is maximum priority of a thread. Value for this is 10.

Get and Set Thread Priority:

public final int getPriority(): java.lang.Thread.getPriority() method returns priority of given thread.
public final void setPriority(int newPriority): java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.
Examples of getPriority() and set

filter_none
edit
play_arrow

brightness_4

```
// Java program to demonstrate getPriority() and setPriority()
import java.lang.*;

class ThreadDemo extends Thread
{
   public void run()
   {
      System.out.println("Inside run method");
   }

   public static void main(String[]args)
   {
      ThreadDemo t1 = new ThreadDemo();
      ThreadDemo t2 = new ThreadDemo();
      ThreadDemo t3 = new ThreadDemo();

      System.out.println("t1 thread priority : " +
                  t1.getPriority()); // Default 5
      System.out.println("t2 thread priority : " +
                  t2.getPriority()); // Default 5
      System.out.println("t3 thread priority : " +
                  t3.getPriority()); // Default 5

      t1.setPriority(2);
      t2.setPriority(5);
      t3.setPriority(8);

      // t3.setPriority(21); will throw IllegalArgumentException
      System.out.println("t1 thread priority : " +
```

```
                    t1.getPriority());  //2
        System.out.println("t2 thread priority : " +
                    t2.getPriority()); //5
        System.out.println("t3 thread priority : " +
                    t3.getPriority());//8

        // Main thread
        System.out.print(Thread.currentThread().getName());
        System.out.println("Main thread priority : "
                + Thread.currentThread().getPriority());

        // Main thread priority is set to 10
        Thread.currentThread().setPriority(10);
        System.out.println("Main thread priority : "
                + Thread.currentThread().getPriority());
    }
}
```

Output:

```
t1 thread priority : 5
t2 thread priority : 5
t3 thread priority : 5
t1 thread priority : 2
t2 thread priority : 5
t3 thread priority : 8
Main thread priority : 5
Main thread priority : 10
```

Note:

Thread with highest priority will get execution chance prior to other threads. Suppose there are 3 threads t1, t2 and t3 with priorities 4, 6 and 1. So, thread t2 will execute first based on maximum priority 6 after that t1 will execute and then t3.
Default priority for main thread is always 5, it can be changed later. Default priority for all other hreads depends on the priority of parent thread.
Example:

filter_none
edit
play_arrow

brightness_4

```
// Java program to demonstrat ethat a child thread
// gets same priority as parent
import java.lang.*;

class ThreadDemo extends Thread
{
```

```
    public void run()
    {
        System.out.println("Inside run method");
    }

    public static void main(String[]args)
    {
        // main thread priority is 6 now
        Thread.currentThread().setPriority(6);

        System.out.println("main thread priority : " +
                Thread.currentThread().getPriority());

        ThreadDemo t1 = new ThreadDemo();

        // t1 thread is child of main thread
        // so t1 thread will also have priority 6.
        System.out.println("t1 thread priority : "
                            + t1.getPriority());
    }
}
```

Output:
Main thread priority : 6
t1 thread priority : 6
If two threads have same priority then we can't expect which thread will execute first. It depends on thread scheduler's algorithm(Round-Robin, First Come First Serve, etc)
If we are using thread priority for thread scheduling then we should always keep in mind that underlying platform should provide support for scheduling based on thread priority.

19 Explain Runnable interface with example?

java.lang.Runnable is an interface that is to be implemented by a class whose instances are intended to be executed by a thread. There are two ways to start a new Thread – Subclass Thread and implement Runnable. There is no need of subclassing Thread when a task can be done by overriding only run() method of Runnable.

Steps to create a new Thread using Runnable :
1. Create a Runnable implementer and implement run() method.
2. Instantiate Thread class and pass the implementer to the Thread, Thread has a constructor which accepts Runnable instance.
3. Invoke start() of Thread instance, start internally calls run() of the implementer. Invoking start(), creates a new Thread which executes the code written in run().
Calling run() directly doesn't create and start a new Thread, it will run in the same thread. To start a new line of execution, call start() on the thread.
Example,

filter_none
edit
play_arrow

brightness_4

```
public class RunnableDemo {

   public static void main(String[] args)
   {
      System.out.println("Main thread is- "
                  + Thread.currentThread().getName());
      Thread t1 = new Thread(new RunnableDemo().new RunnableImpl());
      t1.start();
   }

   private class RunnableImpl implements Runnable {

      public void run()
      {
         System.out.println(Thread.currentThread().getName()
                     + ", executing run() method!");
      }
   }
}
```

Output:

Main thread is- main
Thread-0, executing run() method!
Output shows two active threads in the program – main thread and Thread-0, main method is executed by the Main thread but invoking start on RunnableImpl creates and starts a new thread – Thread-0.

20 Describe InputStream in detail?

Java.io.InputStream Class in Java
InputStream class is the superclass of all the io classes i.e. representing an input stream of bytes. It represents input stream of bytes. Applications that are defining subclass of InputStream must provide method, returning the next byte of input.
A reset() method is invoked which re-positions the stream to the recently marked position.
InputStream
Declaration :

```
public abstract class InputStream
  extends Object
    implements Closeable
```

Constructor :

InputStream() : Single Constructor
Methods:
InputStream Class in Java.

mark() : Java.io.InputStream.mark(int arg) marks the current position of the input stream. It sets readlimit i.e. maximum number of bytes that can be read before mark position becomes invalid.
Syntax :
public void mark(int arg)
Parameters :
arg : integer specifying the read limit of the input Stream
Return :
void
read() : java.io.InputStream.read() reads next byte of data from the Input Stream. The value byte is returned in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.
Syntax :

public abstract int read()
Parameters :

Return :
Reads next data else, -1 i.e. when end of file is reached.
Exception :
-> IOException : If I/O error occurs.
close() : java.io.InputStream.close() closes the input stream and releases system resources associated with this stream to Garbage Collector.
Syntax :
public void close()
Parameters :

Return :
void
Exception :
-> IOException : If I/O error occurs.
read() : Java.io.InputStream.read(byte[] arg) reads number of bytes of arg.length from the input stream to the buffer array arg. The bytes read by read() method are returned as int. If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte.
Syntax :
public int read(byte[] arg)
Parameters :
arg : array whose number of bytes to be read
Return :

reads number of bytes and return to the buffer else, -1 i.e. when end of file is reached.
Exception :
-> IOException : If I/O error occurs.
-> NullPointerException : if arg is null.
reset() : Java.io.InputStream.reset() is invoked by mark() method. It repositions the input stream to the marked position.
Syntax :
public void reset()
Parameters :

Return :
void
Exception :
-> IOException : If I/O error occurs.
markSupported() : Java.io.InputStream.markSupported() method tests if this input stream supports the mark and reset methods. The markSupported method of InputStream returns false by default.
Syntax :
public boolean markSupported()
Parameters :

Return :
true if input stream supports the mark() and reset() method else,false
skip() : Java.io.InputStream.skip(long arg) skips and discards arg bytes in the input stream.
Syntax :
public long skip(long arg)
Parameters :
arg : no. of bytes to be skipped
Return :
skip bytes.
Exception :
-> IOException : If I/O error occurs.
Java Program explaining InputStream Class methods :

filter_none
edit
play_arrow

brightness_4
```
// Java program illustrating the working of InputStream method
// mark(), read(), skip()
// markSupported(), close(), reset()
import java.io.*;

public class NewClass
{
```

```
public static void main(String[] args) throws Exception
{
   InputStream geek = null;
   try {

      geek = new FileInputStream("ABC.txt");

      // read() method : reading and printing Characters
      // one by one
      System.out.println("Char : "+(char)geek.read());
      System.out.println("Char : "+(char)geek.read());
      System.out.println("Char : "+(char)geek.read());

      // mark() : read limiing the 'geek' input stream
      geek.mark(0);

      // skip() : it results in redaing of 'e' in G'e'eeks
      geek.skip(1);
      System.out.println("skip() method comes to play");
      System.out.println("mark() method comes to play");
      System.out.println("Char : "+(char)geek.read());
      System.out.println("Char : "+(char)geek.read());
      System.out.println("Char : "+(char)geek.read());

      boolean check = geek.markSupported();
      if (geek.markSupported())
      {
         // reset() method : repositioning the stram to
         // marked positions.
         geek.reset();
         System.out.println("reset() invoked");
         System.out.println("Char : "+(char)geek.read());
         System.out.println("Char : "+(char)geek.read());
      }
      else
         System.out.println("reset() method not supported.");


      System.out.println("geek.markSupported() supported"+
                  " reset() : "+check);

   }
   catch(Exception excpt)
   {
      // in case of I/O error
      excpt.printStackTrace();
```

```
        }
        finally
        {
            // releasing the resources back to the
            // GarbageCollector when closes
            if (geek!=null)
            {
                // Use of close() : closing the file
                // and releasing resources
                geek.close();
            }
        }
    }
}
```

Note :
This code won’t run on online IDE as no suc file is present here.
You can run this code on your System to check the working.
ABC.txt file used in the code has

HelloGeeks
Output :

Char : H
Char : e
Char : l
skip() method comes to play
mark() method comes to play
Char : o
Char : G
Char : e
reset() method not supported.
geek.markSupported() supported reset() : false