ATHARAV INSTITUTE OF COMPUTER TRAINING AND EDUCATION
Sant Tukaram Nagar,Near Dr.D.Y.Patil college campus Pimpri
Mobile :: 7028851096/7721818535

==========

JAVA

Java Platform

Java platform consist of JVM and implementation of standard libraries.

To develop applications for different types of systems,there are now three types of Platforms.

- **J2SE**
  Java platform – standard edition
  To build applications lime applet and client applications.

- **J2EE**
  Java platform – Enterprise edition
  Used for building server side applications.

- **J2ME**
  Java platform – Micro edition
  It enables the building of java applications for 'micro-devices' that include hand – held devices like phone

==========

Object Oriented Principles

1. Encapsulation

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as as single unit. In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

To achieve encapsulation in Java
Declare the variables of a class as private.
Provide public setter and getter methods to modify and view the variables values.

2. Inheritance

It is a process, in which new class is created by using properties of old class. The old class is called as base or super class and new is called as Sub or derived class.
It is a process in which one object acquires properties of another object.
It is run time process.

==========

3. Polymorphism

Poly means many, morphism means forms. We can have many forms of functions or constructor.

There are two types of Polymorphism :

- Compile time Polymorphism
  Overloading:
  - Method Overloading
    In a class two or more than two methods are defined which share same name with different parameters. (No of parameters or data type of the parameters).
    Return type alone cannot differentiate between two methods.
  - Constructor Overloading:
    Constructor can be overloaded.
  - Run time Polymorphism Overriding

4. Abstraction

In a clasAs per dictionary, Abstraction is the quality of dealing with ideas rather than events. for example when you consider the case of e-mail, complex details such as what happens soon you send an e-mail, the protocol your email server uses are hidden from the user, therefore to send an e-mail you just need to type the content, mention the address of the receiver and click send.

like wise in Object oriented programming Abstraction is a process process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words user will have the information on what the object does instead of how it does it.

In Java Abstraction is achieved using Abstract classes, and Interfaces. s hierarchy methods can be defined with same name with same type signature.

Class

Template that describes the kind of state and behavior that object of its type support. A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- data member
- method
- constructor
- block
- class and interface

==========

Object

At runtime, then the JVM encounters the new keyword, it will use the classes to make an object which is instance of the class. That object will have its own state, and access to all of the behavior defined by class.

State (Instance Variable)

Each object (Instance of class) will have its own unique set of instance variable as defined in the class.

Behavior(Method)

When programmer creates a class. Methods are where the class logic is stored. Methods are where the real work gets done. They are where algorithms are get executed,and data is manipulated.

ClassContains

- **Instance variable**
  Each object contains unique copy of variable.Instance variable in Java
  A variable that is created inside the class but outside the method, is known as instance variable.Instance variable doesn't get memory at compile time.It gets memory at runtime when object(instance) is created.That is why, it is known as instance variable.
- **Constructor**
  Has same name of the class. It is block like a method, but never have return type.
  When memory is allocated by using new keyword, the constructor of the class is immediately get called.
  The constructor is generally used for initialization.
- **Methods**
  It specify particular behavior. It is block of statements which may return value or may not.

Types of Constructor

- Default Constructor / Implicit constructor
  It is created by compiler. It is zero parameterized blank constructor.
- Explicit Constructor
  It is zero parameterized constructor. It may contains statements.
- Parameterized Constructor
  It the parameters are passed to the constructor it is called as parameterized constructor.
- Copy Constructor
  There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.
  There are many ways to copy the values of one object into another in java. They are:

==========

By constructor
By assigning the values of one object into another
By clone() method of Object class

When the object is passed as a parameter to the constructor it is called as copy constructor.

==========

Advantage of OOPs over Procedure-oriented programming language

	Procedure Oriented Programming	**Object Oriented Programming**
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.

==========

==========

Q What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

Java Naming conventions

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

CamelCase in java naming conventions

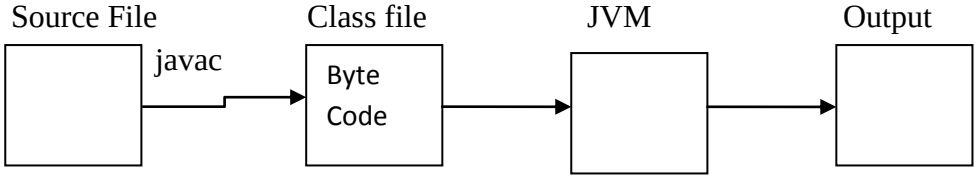
Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

Java Virtual Machine:

It is an interpreter (software).When java source file compiled using compiler (javac), the .class files are generated. These .class files contains byte code which are only understood by JVM. The JVM converts these byte codes into machine code which is understood by hardware and executes the program.

JVM has garbage collector which destroys the objects those have no references(null referances)



==========

Expain the reason for each keyword of public static void main(String args[])?

public- main(..) is the first method called by java environment when a program is executed so it has to accessible from java environment. Hence the access specifier has to be public.

static: Java environment should be able to call this method without creating an instance of the class , so this method must be declared as static.

void: main does not return anything so the return type must be void

The argument String indicates the argument type which is given at the command line and arg is an array for string given during command line.

==========

Garbage Collection:

An object which is no longer needed by the system is automatically removed by JVM (by Garbage Collector). The JVM automatically removes the objects those have no references (null references). You can manually release memory by calling.

```
System.gc();
Or
Runtime.gc();
```

Finalizer

A few situation exists in which a class need to clean itself up before garbage collection. It can do this by implementing a finalizer method. Finallizer are called just before a class is garbage collected. Finalizer are typically used to close/ open files or connections or ensures that related tasks are completed before object is destroyed.

To create a finalizer,simply defines the method called finalize.

```
protected void finalize() throws Throwable
{
   // free resources here
}
```

==========

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By factory method etc.

==========

Annonymous object

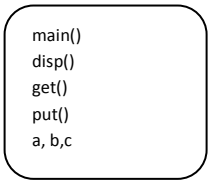
Annonymous simply means nameless. An object that have no reference is known as annonymous object. If you have to use an object only once, annonymous object is a good approach.

The stack and heap

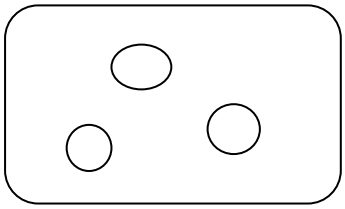
- Java has two areas of memory we care about the heap and the stack.
- Instance variable are variables declared inside the class but outside the method body.
- Local variables are variables declared inside a method or method parameter.
- All local variable lives on the stack, in the frame corresponding to the method where the variables are declared.
- Object reference variables work just like primitive variables – if the references is declared as a local variable, it goes on the stack.
- All objects live in the heap, regardless the reference is local or instance variable.

Where things live.

The stack :Where method invocations and local variable lives.



The Heap
Where all objects live also known as "The Garbage Collective Heap".

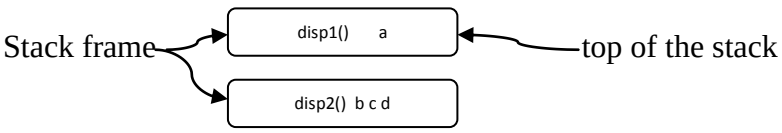


Local variable are declared inside a method, including method parameters. They are temporary, and live only as long as the method is on the stack.

Instance variable are declared inside the class but not inside the method. They live inside the object they belongs to.

- When you call a method, the method lands on the top of a call stack. The new things that's actually pushed on the stack is the stack frame, and it holds the state of the method including which line of code is executing, and the values of all variables.

- The method at the top of the stack is always the currently –running method for the stack. A method stays on the stack until the method hits closing curly braces.



==========

← Local variables (including parameter b)

How much space?

Instance variables lives on the heap, inside the object they belongs to.

Values of an object's instance variables live inside the objects.

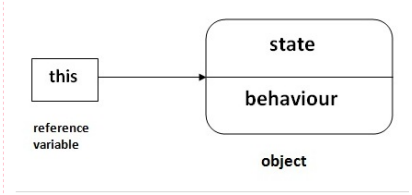
If instance variables are all primitives. Java makes space for all instance variables based on the primitives type. An int need 32 bits and long 64 bits etc. the bit size of an int is the same (32 bits) whether the value of the int is 32,000,000 or 32.

==========

Usage of java this keyword

Here is given the 6 usage of java this keyword.

- this keyword can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this keyword can also be used to return the current class instance.



The this keyword can be used to refer current class instance variable.
If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

Use of this keyword

It can be used inside the M*ethod* or *constructor* of Class. It(***this)*** works as a reference to the current Object whose Method or constructor is being invoked. The ***this*** keyword can be used to refer to any member of the current object from within an instance Method or a constructor.

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

==========

Note:

this keyword can only be the first statement in Constructor.
A constructor can have either this or super keyword but not both.

String class

String represents fixed-length, immutable character sequences.The strings within objects of type String are unchangeable means that the contents of the String instance cannot be changed after it has been created.

Consturctor:

- String()
- String(char[])
- String(byte [])
- String(String object)

Important Methods:

- int length( )
  The length of a string is the number of characters that it contains.
- public byte[] getBytes(String charsetName)
  Encodes (String) into a sequence of bytes.
- String toString( )
  return a String object that contains the human-readable string that appropriately describes an object of your class
- char charAt(int *where*)
  Extract a single character from a String, *where* is the index of the character that you want to obtain
- boolean equals(Object *str*)
  It returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is case-sensitive.
- boolean equalsIgnoreCase(String *str*)
  It returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is not case-sensitive.
- int compareTo(String *str*)
  Here, *str* is the String being compared with the invoking String. The result of the comparison
  is returned and is interpreted, it returns
  Less than zero The invoking string is less than str.
  Greater than zero The invoking string is greater than str.
  Zero The two strings are equal.

==========

- String substring(int *startIndex*)

  It returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is case-sensitive.
- String substring(int *startIndex*, int *endIndex*)

  Here, *startIndex* specifies the beginning index, and *endIndex* specifies the stopping point.

  The string returned contains all the characters from the beginning index, up to, but not

  including, the ending index.
- String concat(String *str*)

  This method creates a new object that contains the invoking string with the contents

  of *str* appended to the end. concat( ) performs the same function as +.

- String replace(char *original*, char *replacement*)

  Here, *original* specifies the character to be replaced by the character specified by *replacement*.
- String trim( )

  The trim( ) method returns a copy of the invoking string from which any leading and trailing

  whitespace has been removed.
- String toLowerCase( )

  The method toLowerCase( ) converts all the characters in a string from uppercase to Lower case.
- String toUpperCase( )

  The method toLowerCase( ) converts all the characters in a string from lowercase to uppercase.

==========

StringBuffer Class

StringBuffer represents growable and writeable character sequences.

StringBuffer is synchronized that isStringBuffer is thread safe .

Constructors:

- StringBuffer( ) // reserves room for 16 characters
- StringBuffer(int *size*)
- StringBuffer(String *str*)
- StringBuffer(CharSequence *chars*)

Important Methods

- int length( )
- int capacity( )
- StringBuffer append(String *str*)

==========

- StringBuffer insert(int *index*, String *str*)
- StringBuffer reverse( )
- StringBuffer delete(int *startIndex*, int *endIndex*)
- StringBuffer deleteCharAt(int *loc*)
- StringBuffer replace(int *startIndex*, int *endIndex*, String *str*)
- String substring(int *startIndex*)
- String substring(int *startIndex*, int *endIndex*)

==========

Differences:

Index	*String*	*String Buffer*	*String Builder*
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Thread Safe	Fast	Very slow	Fast

==========

StringTokenizer class (java.util package)

The string tokenizer class allows an application to break a string into tokens.

Constructor:

public StringTokenizer(String str)

Constructs a string tokenizer for the specified string.

Important Methods:

- public boolean hasMoreTokens()

  Tests if there are more tokens available from this tokenizer's string.

- public String nextToken()

  Returns the next token from this string tokenizer.

==========

==========

StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important Constructors of StringBuilder class

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	is used to reverse the string.
public int capacity()	is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.
public String substring(int	is used to return the substring from the specified

==========

beginIndex)	beginIndex.
public String substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

==========

Wrapper classes

One of the eight classes of java.lang package are known as wrapper class in java. Java provides specialized classes corresponding to each of the primitive data types. These are called wrapper classes.

Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these reasons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

The list of eight wrapper classes are given

Primitive	Wrapper	Primitive	Wrapper
boolean	Boolean	int	Integer
char	Character	long	Long
byte	Byte	float	Float
short	Short	double	Double

Wrapper class Example: Primitive to Wrapper

```
public class WrapperExample1
{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
System.out.println(a+" "+i+" "+j);
}}
```

Output:

==========

20 20 20

Wrapper class Example: Wrapper to Primitive

```
public class WrapperExample2
{
 public static void main(String args[])
 {
   //Converting Integer to int
    Integer a=new Integer(3);
     int i=a.intValue();//converting Integer to int
     int j=a;   //unboxing, now compiler will write a.intValue() internally

 System.out.println(a+" "+i+" "+j);
 }}
```

Output: 3 3 3

Character_class

Consturctor

Character(char *ch*)

Important Methods:

- public static boolean isDigit(char)
- public static boolean isLetter (char)
- public static boolean isLetterOrDigit(char)
- public static boolean isWhitespace(char)
- public static boolean i isUpperCase (char)
- public static boolean isLowerCase (char)
- public static char toLowerCase (char)
- public static char toUpperCase (char)

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation{
 int data=50;
   void change(int data)
{
   data=data+100;//changes will be in the local variable only
```

```
==========================================================================
    }
       public static void main(String args[])
      {
       Operation op=new Operation();
       System.out.println("before change "+op.data);
        op.change(500);
       System.out.println("after change "+op.data);
   }
  }
```

Output:before change 50
after change 50

==========

Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.
Legal code for strictfp keyword

The strictfp keyword can be applied on methods, classes and interfaces.

```
strictfp class A{}//strictfp applied on class
strictfp interface M{}//strictfp applied on interface

class A{
strictfp void m(){}//strictfp applied on method
}
```

Illegal code for strictfp keyword

The strictfp keyword cannot be applied on abstract methods, variables or constructors.

```
class B
{
  strictfp abstract void m();//Illegal combination of modifiers
}
class B
{
  strictfp int data=10;//modifier strictfp not allowed here
}
```

==========

```
class B
{
    strictfp B()
    {}   //modifier strictfp not allowed here
 }
```

==========

Creating API Document | javadoc tool

We can create document api in java by the help of javadoc tool. In the java file, we must use the documentation comment /**... */ to post information for the class, method, constructor, fields etc.

Let's see the simple class that contains documentation comment.

```
package com.abc;
/** This class is a user-defined class that contains one methods cube.*/
public class M
{
   /** The cube method prints cube of the given number */
   public static void  cube(int n){System.out.println(n*n*n);}
 }
```

To create the document API, you need to use the javadoc tool followed by java file name. There is no need to compile the javafile.

On the command prompt, you need to write:

javadoc M.java

to generate the document api. Now, there will be created a lot of html files. Open the index.html file to get the information about the classes.

==========

Runtime class

The Runtime class encapsulates the run-time environment. You cannot instantiate a Runtime object. However, you can get a reference to the current Runtime object by calling the static method Runtime.getRuntime( ).

Important Methods:

- void gc( ) Initiates garbage collection.
- static Runtime getRuntime( ) Returns the current Runtime object.

==========

System class

==========

The System class holds a collection of static methods and variables. The standard input,output, and error output of the Java run time are stored in the in, out, and err variables.

Important Method:

static void gc( )

Initiates garbage collection.

==========

Object class

Object is a superclass of all other classes.

Important Methods:

- Object clone( )
  Creates a new object that is the same as the invoking object.
- boolean equals(Object object)
  Returns true if the invoking object is equivalent to object.
- protected void finalize( )
  Default finalize( ) method. This is usually overridden by subclasses.
- final void notify( ) Resumes execution of a thread waiting on the invoking object.
- final void notifyAll( )
  Resumes execution of all threads waiting on the invoking object.
- String toString( ) Returns a string that describes the object.
- final void wait( ) throws InterruptedException
  Waits on another thread of execution.
- final void wait(long milliseconds) throws InterruptedException
  Waits up to the specified number of milliseconds on another thread of execution.

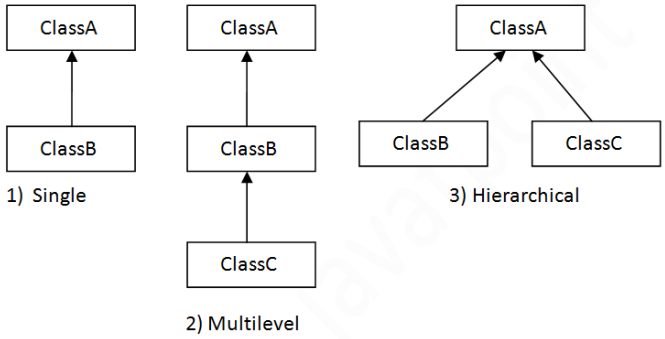
- Object getClass( )
  It returns a Class object that corresponds to the type of the class of the object on which it is called.

==========

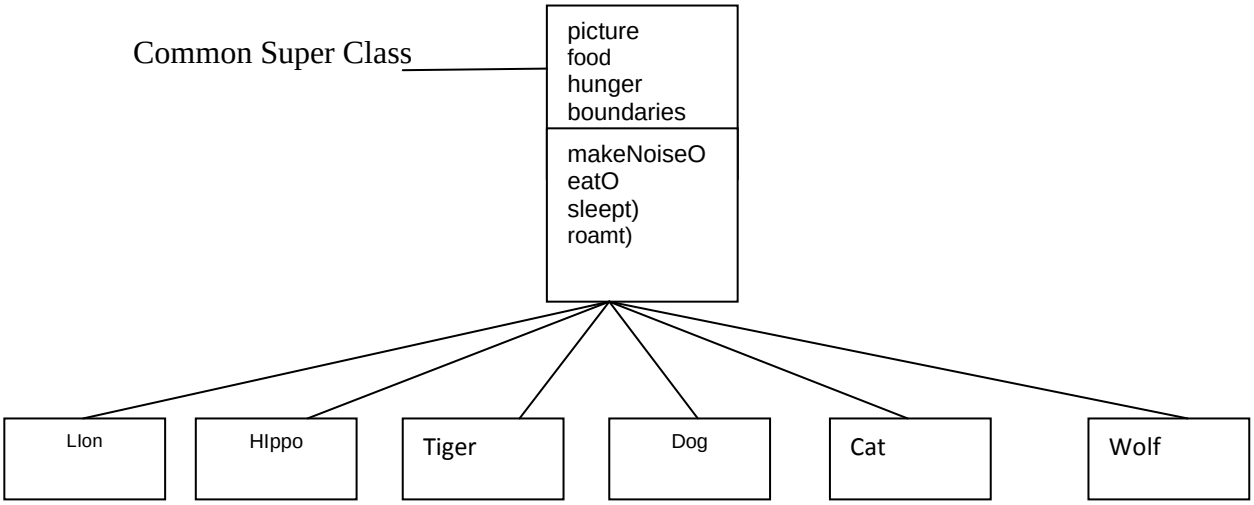
==========

Inheritance

It is a process, in which new class is created by using properties of old class. The old class is called as base or super class and new is called as Sub or derived class.
It is a process in which one object acquires properties of another object.
It is a run time process.
The extends keyword is used to inherits the properties from class.
One interface acquires the properties of another interface.
Types of inheritance



Using inheritance to avoid duplicating code in sub classes.



- A subclass extends a superclass.
- A subclass inherits all public instance variables and methods of the superclass, but does not inherit's the private instance variable and methods of the superclass.
- Inherited methods can be overridden, instance variables cannot be overridden (although they can be redefined in the subclass, but that's not the same thing, and there's almost never need to do it)

==========

- Use the Is-A test to verify that your inheritance hierarchy is valid. If X extend Y, then X Is-A Y must make sense.
- The Is-A relationship works in only one direction. A Hippo is an animal, but not all animals are Hippos.
- When method is overridden in a sub-class, and the method is invoked on the instance of the subclass, the overridden version of the method is called.
- If class B extends A and C extend B, class B Is-A class A, class C Is-A class B and class C also Is-A class A

==========

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 same method and you call it from child class object, there will be ambiguity to call 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 now.

==========

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee
{
    int id;
    String name;
     Address address;   //Address is a class
  ...
  }
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Q Why use Aggregation?

For Code Reusability.

==========

Q When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Use of super keyword

It is used to access the members of super class. e.g. instance variable and methods

1. super.variableName;
   Super class variables can be accessed with super keyword.
2. super.method();
   Super class methods can be accessed/called with super keyword.
3. super()
   Super class constructor can be called with super keyword. By default or implicitly constructor of super get is called.

Non Access Member Modifier

1) abstract

a) abstract class
Abstract class never be instantiated.
Note: The class which is not abstract is called as concrete class.

b) abstract method
Abstract method ends with semicolon. It should be overridden in non abstract class. When method is declared as abstract then class should be declared as abstract.
Note: The method which is not abstract is called as concrete method.

A factory method is the method that returns the instance of the class.

2) final

a) final class [MustInherit]
Cannot create sub class of final class.

b) final variable [constant]
Cannot re-assign the value to final variable. It must initialize a final variable when it is declared.

c) final method [MustNotOveride]
Prevent a method from being overridden in a sub class.

3) static

a) static class
It should be inner class.

b) static variable

==========

Instance variables declared as static are, essentially, global variables. All objects refers same copy of static variable. Can be accessed using name of the class.

c) static method

static methods can be accessed without instantiating the class.

Methods declared as static have several restrictions:

• They can only call other static methods.
• They must only access static data.
• They cannot refer to this or super in any way.

- static block
  - Static block is mostly used for changing the default values of static variables.
  - This block gets executed when the class is loaded in the memory.
  - A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.
  - Static block in java is executed before main method. It is mostly used in JDBC.
  - static block get executed first time before constructor and when you instantiate the class first time.

Note:

- static methods and variables can be accessed without instantiating the class. You need to only specify the name of their class followed by the dot operator

- static methods and variables can be used independently of any object.

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

Example of blank final variable

```
class Student{
    int id;
    String name;
   final String PAN_CARD_NUMBER;
    ...
    }
```

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

==========

Q) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit; //blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
  }
 }
```

Q) static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
  }
 }
```

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Q) Can we override static method?

No, static method cannot be overridden.

Q) Why we cannot override static method?

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

==========

Q) Can we override java main method?

No, because main is a static method.

Instance initializer block:

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.

The initialization of the instance variable can be directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

Que) What is the use of instance initializer block while we can directly assign a value in instance data member? For example:

```
class Bike{
   int speed=100;
}
```

Why use instance initializer block?

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

What is invoked firstly instance initializer block or constructor?

NO ,Instance initializer block is invoked after constructor. Instance initializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super().

Rules for instance initializer block :

There are mainly three rules for the instance initializer block. They are as follows:

The instance initializer block is created when instance of the class is created.

The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).

The instance initializer block comes in the order in which they appear.

==========

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

static binding (also known as early binding).

dynamic binding (also known as late binding).

Understanding Type

Let's understand the type of instance.

1) variables have a type

Each variable has a type, it may be primitive and non-primitive.

```
int data=30;
```

Here data variable is a type of int.

==========

2) References have a type

```
class Dog{
        public static void main(String args[]){
        Dog d1;//Here d1 is a type of Dog
        }
}
```

3) Objects have a type

An object is an instance of particular java class,but it is also an instance of its superclass.

```
class Animal{}

class Dog extends Animal{
 public static void main(String args[]){
  Dog d1=new Dog();
 }
}
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example of static binding

```
class Dog
 {
        private void eat()
        {
           System.out.println("dog is eating...");
        }
        public static void main(String args[])
        {
                Dog d1=new Dog();
                d1.eat();
        }
 }
```

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

==========

Example of dynamic binding

```
 class Animal
{
    void eat()
    {System.out.println("animal is eating...");}
 }
     class Dog extends Animal
{
 void eat()
 {System.out.println("dog is eating...");}

 public static void main(String args[])
 {
  Animal a=new Dog();
  a.eat();
 }     }
```

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So compiler doesn't know its type, only its base type.

==========

Reference Equality

Two references that refers to the same object on the heap are equals.

== operator is used to check equality.

Object Equality

Each object has unique hashcode value. Two equivalent objects return same hash code.

1) Method of Object class

```
public boolean equals(Object obj)
 {
        return (this == obj);
 }
```

The equals method for class Object implements the most discriminating possible equivalence relation on objects, that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object x == y has the value true.

Note : It is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.

2) public int hashCode()

   Returns a hash code value for the object. This method is supported for the benefit of hashtables. As much as is reasonably practical, the hashCode method defined by class

==========

Object does return distinct integers for distinct objects.

==========

Java instanceof

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

```
class Dog2{
 public static void main(String args[])
{
        Dog2 d=null;
        System.out.println(d instanceof Dog2);//false
 } }
```

==========

Interface

A java interface solves the problem of multiple inheritance .

A java interface is like 100% pure abstract class.

Interface is defined with interface keyword.

Syntax::

```
interface InterfaceName
{
        public static final variables.
        public abstract methods
}
```

Interface variables are implicitly / by default are public static final.

Interface methods are implicitly / by default are public abstract.

implements keyword is used to implement interface.

The class that implements an interface must implement **all the methods** of the interface, since all methods are public abstract.

The class can implements multiple interface.

Q) Why use Java interface?

There are mainly three reasons to use interface. They are given below.

It is used to achieve fully abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

Q) Multiple inheritance is not supported through class in java but it is possible by interface, why?

==========

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Interface inheritance
A class implements interface but one interface extends another interface .

Q) What is marker or tagged interface?
An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
//How Serializable interface is written?
public interface Serializable{
}
```

Nested Interface in Java
Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

```
interface printable
{
 void print();
   interface MessagePrintable
   {
     void msg();
   }
}
```

What happen when

1) There are more than one interfaces, and these are implemented in that class.
   All the methods of all the interfaces should be overridden in that class.
2) First interface extends to second interface. Both have different methods. And second interface is implemented in a class.
   If the second interface inherits the properties of first interface. All the methods of second interface (including inherited methods) should be overridden in that class.
3) First interface extend to second interface. Both have methods with same name with same parameter list. Then you need to implements only one method for them.
4) If the second interface inherits the properties of first interface. All the methods of second interface (excluding inherited methods) should be overridden in that class.
   Only one method should be overridden which has same name with same parameter list.

==========

5) How to call the methods of interface with interface object.
Just pass the reference of sub class to reference variable of interface. Now we can call the methods (access the members) of interface with interface object.

Difference between abstract class and interface

Abstract class	**Interface**
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Java 8 : We can define methods inside methods which are default or static
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can have static methods, main method and constructor.	Interface can't have method except default and static. It cant have constructor.
5) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
6) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

7) Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Cloneable interface

Class Object is the root of the class hierarchy.

Every class has Object as a super class. All objects, including arrays, implement the methods of this class.

Cloning

Cloning creates an identical copy of an object.

To clone an object of a class, the class must implements interface. This copy must implement Cloneable interface. This is an empty interface (tagging or marking interface).

To clone objects of a class, override the clone() method of the Object class.

Q Why use clone() method ?

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing to be performed that is why we use object cloning.

Q) Advantage of Object cloning

==========

Less processing task.

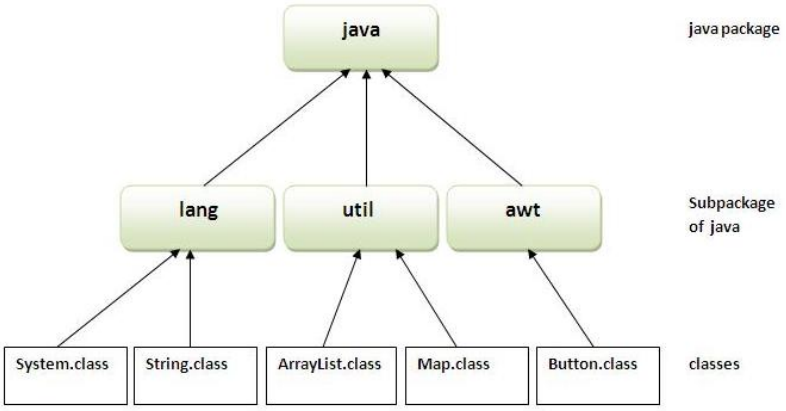
==========

JavaPackage

A java package is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
Here, any Single number of classes ,it may or may not extends each other.
Class in package can extends each other.
Generally, similar type of classes are placed in single package.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.



package keyword is used to define package.

```
package packagename;
```

Import package

```
import packagename;
```

Some build-packages.
java.io
java.util
By default package included :: java.lang

Q. How to access package from another package?

==========

There are three ways to access the package from outside the package.

import package.*;
import package.classname;
fully qualified name.

1) Using packagename:

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

3**) Using fully qualified name**

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

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

Class B

```
package mypack;
class B
{
  public static void main(String args[])
{
    pack.A obj = new pack.A();//using fully qualified name
     obj.msg();
  }
}
```

Note: Sequence of the program must be package then import then class.

1. Package

==========

2. Import
3. Class

Q. What about package class?

The package class provides methods to get information about the specification and implementation of a package. It provides methods such as getName(), getImplementationTitle(), getImplementationVendor(), getImplementationVersion() etc.

Static Import:

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

```
import static java.lang.System.*;
class StaticImportExample{
  public static void main(String args[]){

    out.println("Hello");//Now no need of System.out
    out.println("Java");
  }      }
```

Access Specifiers :

Access specifier for class :: default and public

	Private	No Modifier	Protected	Public
Same class	Y	Y	Y	Y
Same package subclass	N	Y	Y	Y
Same package non-subclass	N	Y	Y	Y
Different package subclass	N	N	Y	Y
Different package	N	N	N	Y

==========

non-subclass				

Encapsulation in Java

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.


Capsule

The Java Bean class is the example of fully encapsulated class.

Advantage of Encapsulation in java

By providing only setter or getter method, you can make the class read-only or write-only.

It provides you the control over the data. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

```
package com.javatpoint;
public class Student
{
        private String name;
        public String getName()
        {
                return name;
        }
        public void setName(String name)
        {
                this.name=name
        } }

  //save as Test.java
  package com.javatpoint;
  class Test
 {
        public static void main(String[] args)
       {
                Student s=new Student();
                s.setname("Ranjit");
                System.out.println(s.getName());
         }     }
```

==========
==========

Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location. Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.
Array in java is index based, first element of the array is stored at 0 index.
java array

Advantage of Java Array

Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.
Random access: We can get any data located at any index position.

Disadvantage of Java Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

Single Dimensional Array
Multidimensional Array

What is the class name of java array?

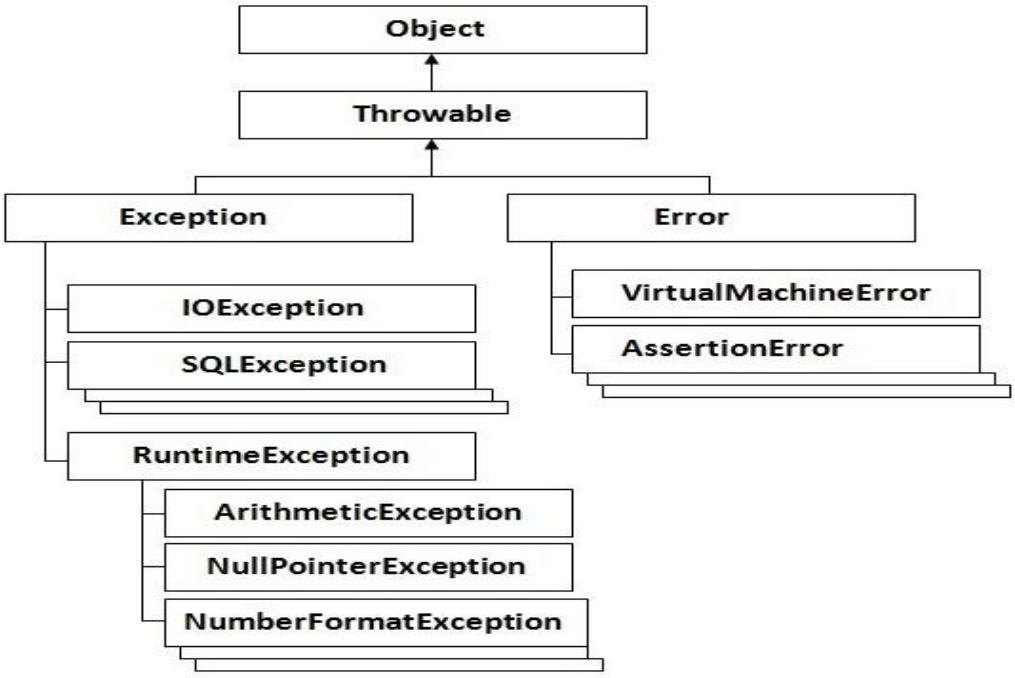
In java, array is an object. For array object, an proxy class is created whose name can be obtained by getClass().getName() method on the object.

==========

Exception Handling:

Exception are objects. The exception is an error arises at runtime. It is an abnormal situation arises due to bad / wrong input.
Exception classes are sub-classes of Exception. This class is derived from class Throwable.

==========



Classes that derives from Error represents unusual situations that are not caused by program error, and indicate things that would not normally happen during program execution,such as JVM running out of memory. Generally,your application won't be able to recover from an Error, So you are not required to handle them.

Exception is handled to avoid illegal termination of the program. Exception can be handled by using five keywords these are try, catch, finally, throws and throw.

- try block
  The objects of exceptions are thrown from this block, if exception arises.
- catch block
  The exception thrown by try block are catch in this block.
- Finally
  It executes when exceptions are thrown or not.
- Throws
  Generally used to handle exceptions of built in exceptions(Classes)
- throw
  Used to handle user defined exceptions.

Points: There are two types of exceptions

1) **Checked Exception**
   The complier forces to handle it compulsory.

==========

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.
E.g. IOException, InterruptedException, ClassNotFoundException, SQLException,
FileNotFoundException

2) **Unchecked Exception**
The complier does not force to handle it compulsory.
The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
They simply occurs at run time
e.g.
ArrayIndexOutOfBoundsException
NumberFormatException
NegativeArraySizeException
NullPointerException
ClassCastException
ArithmeticException

- A method can throw an exception when something fails at runtime .
- An exception is always an object of type exception.(which, as you remember from the polymorphism chapter means the object is from a class that has exception somewhere up its inheritance tree.)
- The compiler does not pay attention that are of type Runtime –Exception. A Runtime –Exception does not have to be declared or wrapped in a try/catch(although you're free to do either or both of those things).
- All Exception the compiler cares about are called 'checked exception which really means compiler-checked exception. Only Runtime-Exception are excluded from compiler checking. All other exception must be acknowledged in your code, according to the rules.
- A methods throws an exception with the keyword throw, followed by new exception object:
  throw new Demo()
- Methods that might throw a checked exception must announce it with a throws exceptions declarations.
- If your code calls a checked exception throwing method, it must reassure the complier that precaution have been taken.

==========

- If you are prepared to handle the exception, wrap the call in a try / catch and put your exception handling / recovery code in the catch block.

==========

Java Inner Class

Java inner class or nested class is a class i.e. declared inside the class or interface.
We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
Additionally, it can access all the members of outer class including private data members and methods.
Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:
1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
3) Code Optimization: It requires less code to write.

Q What is the internal code generated by the compiler for member inner class ?

What are the two ways to create annonymous inner class ?
Can we access the non-final local variable inside the local inner class ?
How to access the static nested class ?
Can we define an interface within the class ?
Can we define a class within the interface ?

Q Difference between nested class and inner class in Java ?

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

1. Non-static nested class(inner class)

a)Member inner class

==========

b)Annomynous inner class
c)Local inner class

2. Static nested class

Type	Description
Member Inner	A class created within class and outside method.
Anonymous Inner	A class created for implementing interface or extending class. Its name is
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

1. Member Inner Class

```
class Outer
{
  //code
   class Inner
   {
     //code
   }
}
```

2. Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

Class (may be abstract or concrete).
Interface

Example:

```
abstract class Person {
  abstract void eat();
}
class TestAnonymousInner
{
 public static void main(String args[])
 {
   Person p=new Person(){
   void eat(){System.out.println("nice fruits");
 }       };
  p.eat();
```

==========

```
    }      }
```

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

3. Java Local inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Java local inner class example

```
public class localInner1{
 private int data=30;//instance variable
 void display(){
  class Local{
   void msg(){System.out.println(data);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner1 obj=new localInner1();
  obj.display();
 }      }
```

4. Java static nested class

A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

It can access static data members of outer class including private.

Static nested class cannot access non-static (instance) data member or method.

==========

Java static nested class example with instance method

```
class TestOuter1{
  static int data=30;
  static class Inner{
   void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter1.Inner obj=new TestOuter1.Inner();
  obj.msg();
  }  }
```

Output:

data is 30

Java Nested Interface

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.

Nested interfaces are declared static implicitely.

Syntax of nested interface which is declared within the interface

```
interface interface_name{
...
 interface nested_interface_name{
  ...
 }  }
```

Syntax of nested interface which is declared within the class

```
class class_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

==========

==========

Multithreading

Introduction
Defination:
A multutithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread,and each thread define a separate path of execution.Thus,multithreading is specialized form of multitasking.

The human body performs a great variety of operations in parallel—or, as we will say throughout this chapter, concurrently. Respiration, blood circulation, digestion, thinking and walking, for example, can occur concurrently. All the senses—sight, touch, smell, taste and hearing—can be employed at once.
Computers, too, can perform operations concurrently. It is common for personal computers to compile a program, send a file to a printer and receive electronic mail messages over a network concurrently. Only computers that have multiple processors can truly execute multiple instructions concurrently.
Operating systems on single-processor computers create the illusion of concurrent execution by rapidly switching between activities, but on such computers only a single instruction can execute at once. Respiration, blood circulation, digestion, thinking and walking, for example, can occur concurrently.

- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a *thread,* and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

Multitasking

- Process based multitasking:
  In it more than one process run concurrently.
- Thread based multitasking:
  In it multiple threads executes concurrently.

1) Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

Note:

- Multithreading in java is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing.

==========

- Multiprocessing and multithreading, both are used to achieve multitasking.
- Each thread has its own method-call stack and program counter, allowing it to execute concurrently with other threads.
- Threading is a facility to allow multiple activities to coexist within a single process. Most modern operating systems support threads, and the concept of threads has been around in various forms for many years.\
- Every Java program has at least one thread -- the main thread. When a Java program starts, the JVM creates the main thread and calls the program's main() method within that thread.

Advantage of Java Multithreading

- It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- You can perform many operations together so it saves time.
- Threads are independent so it doesn't affect other threads if exception occur in a single thread.

Creating threads:

- *Using subclass of the Thread class*
- *Implementing the Runnable interface*

API

1. **Runnable interface**
   public abstract void run();

2. **public class Thread implements Runnable**
   Constructors:
   - Thread()
   - Thread(java.lang.Runnable)
   - Thread(java.lang.String)
   - Thread(java.lang.Runnable, java.lang.String)

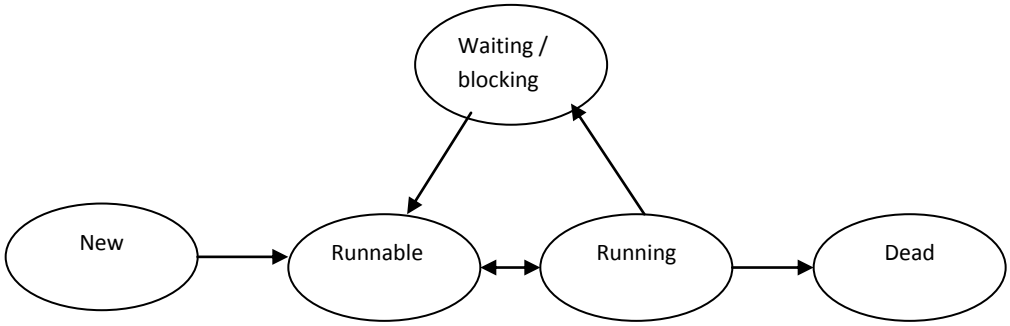
   Important methods
   a) void start()
   b) public void run()
   c) final void stop();
   d) public final native boolean isAlive()
   e) public final void suspend()
   f) public final void resume()
   g) public final void setPriority(int)
   h) public final int getPriority()
   i) public final void setName(java.lang.String)
   j) public final java.lang.String getName()
   k) public final synchronized void join(long) throws InterruptedException

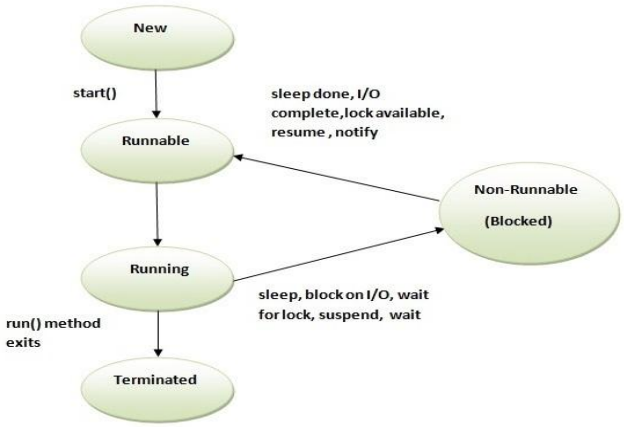
==========

l) Thread currentThread()
m) void yield()
n) void sleep(long) throws java.lang.InterruptedException

==========

Threads State





Threads State

New-

This is the state the thread is in after the thread instance has been created, but the start () method has not been invoked on the thread. At this point thread is considered not alive.

Runnable-

This is the state is in when it's eligible to run, but the scheduler has not selected it to be running thread.

A thread first enters the runnable state when the start () method is invoked, but a thread can also return to the runnable state after either running or coming back from a blocked , waiting or sleeping state.

When the thread is in runnable state it is considered alive.

Running-

This is the state a thread is in when thread scheduler selects it (from the runnable pool) to be the

==========

currently executing process.

Waiting/Blocked/Sleeping-

This is the state is in when it's not eligible to run, but it might return to runnable state later if a particular event occurs.

Dead-

A thread is considered dead when its run () method completes.Once a thread is dead, it can never be brought back to life.

Thread Scheduler:

is a part of the JVM that decides which thread should run at any given moment and also takes the thread out of the run state.

The order in which runnable threads are chosen to run is not guaranteed.

The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Q) Difference between preemptive scheduling and time slicing

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

Advantage of Java Thread Pool

Better performance It saves time because there is no need to create new thread.

Real time usage

It is used in Servlet and JSP where container creates a thread pool to process the request.

Queue behavior

is typical, it is not guaranteed.Queue behavior means that when a thread has finished with its turn it moves to the end of the line of the runnable pool and waits until it eventually gets to the front of the line, where it can be chosen again.

==========

In fact we call the runnable pool, rather than a runnable queue.So not guaranteed order.

Q) Can we start a thread twice ?

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

Q) What if we call run() method directly instead start() method?

Each thread starts in a separate call stack.

Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

The join() method:

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Thread Priority:-

The JVM will never change a thread's proirity. Default priority is 5.

Can be set between 1 - 10.
static final variables (Constants)
Thread.MIN_PRIORITY (1)
Thread.NORM_PRIORITY (5)
Thread.MAX_PRIORITY (10)

Only to improve the efficiency of the program the prority of the thread given.

Shutdown Hook

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

When does the JVM shut down?

The JVM shuts down when:

- user presses ctrl+c on the command prompt
- System.exit(int) method is invoked
- user logoff
- user shutdown etc.

==========

The addShutdownHook(Runnable r) method

The addShutdownHook() method of Runtime class is used to register the thread with the Virtual Machine. Syntax:

public void addShutdownHook(Runnable r){}

The object of Runtime class can be obtained by calling the static factory method getRuntime().

For example:

```
Runtime r = Runtime.getRuntime();
```

Factory method

The method that returns the instance of a class is known as factory method.

Simple example of Shutdown Hook

```
class MyThread extends Thread{
   public void run(){
      System.out.println("shut down hook task completed..");
   }
}

public class TestShutdown1{
public static void main(String[] args)throws Exception {

Runtime r=Runtime.getRuntime();
r.addShutdownHook(new MyThread());

System.out.println("Now main sleeping... press ctrl+c to exit");
try{Thread.sleep(3000);}catch (Exception e) {}
}
}
```

Synchronization in Java

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

The synchronization is mainly used to

- To prevent thread interference.
- To prevent consistency problem.

There are two ways for synchronization:

==========

1) **Synchronized Method**
2) **Synchronized block**

The difference is in which lock is being acquired:

- synchronized method acquires a method on the whole object. This means no other thread can use any synchronized method in the whole object while the method is being run by one thread.
- synchronized blocks acquires a lock in the object between parentheses after the synchronized keyword. Meaning no other thread can acquire a lock on the locked object until the synchronized block exits.

Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.

Thread-Safe classes-

When a class has been carefully synchronized to protect its data, we say the class is " thread-safe".

Many classes in the Java API already use synchronization internally in order to make the class thread-safe.

For eg.-

StringBuffer and StringBuilder are nearly identical classes, except that all the methods in StringBuffer are synchronized when necessary, while those in StringBuilder are not. Generally, this makes StringBuffer safe to use in multithreaded environment, while StringBuilder is not.

==========

Daemon threads

Java program exits when all of its threads have completed, but this is not exactly correct. What about the **hidden system threads**, such as **the garbage collection thread** and others created by the JVM? We have no way of stopping these. If those threads are running, how does any Java program ever exit?

Daemon Thread in Java

Daemon thread in java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

==========

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.
Points to remember for Daemon Thread in Java

It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.

Its life depends on user threads.

It is a low priority thread.

Q) Why JVM terminates the daemon thread if there is no user thread?
The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.
Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

No.	Method	Description
1)	public void setDaemon(boolean status)	is used to mark the current thread as daemon thread or user thread.
2)	public boolean isDaemon()	is used to check that current is daemon.

Deadlock in java
Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for anobject lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

==========

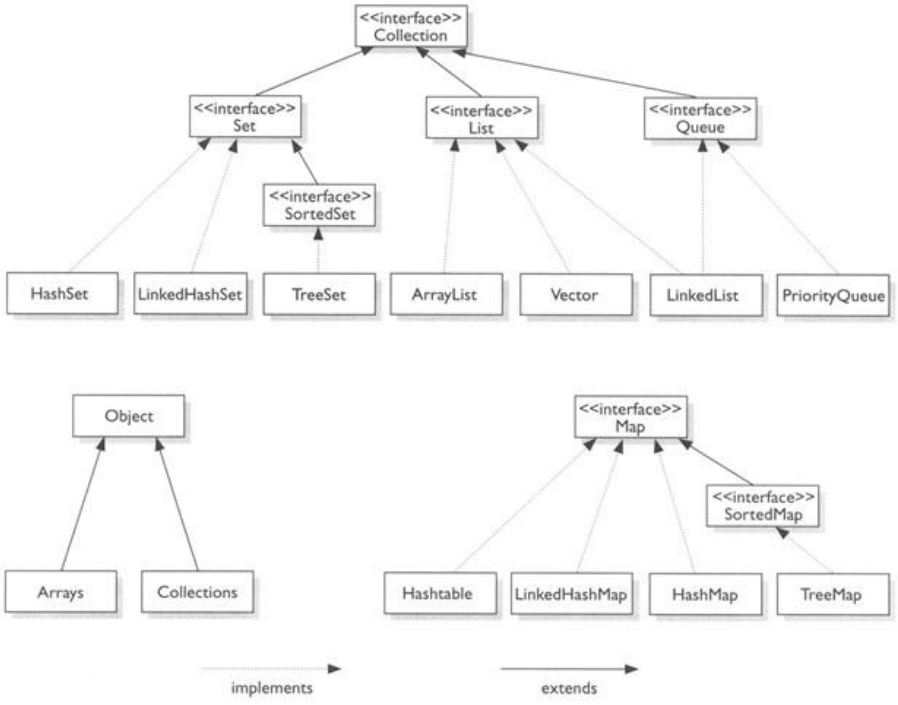
==========

Collection in Java

- Sometimes called a container- is simply an object that group multiple elements into a single unit.
- Collection are used to store, retrieve, mainipulate and communicate aggregate data.
- The core collection interfaces encapsulate different types of collections, which are shown in the figure below.[**Collection interfaces**]

Autoboxing with Collections

The collection can hold objects but not primitives. Autoboxing handles the wrapping for us.



---

1. Collection:
   The root of the collection hierarchy. A collection represents a group of objects known as elements. The java platform doesn't provide any direct

implementations of this interface but provides implementations of more specific sub interfaces,such as Set and List.

2. Set
   A collection that **can not** contain **duplicate elements**.
3. List
   An ordered collection (Sequence). Lists **can** contain duplicate elements.
4. Queue:
   A collection used to hold multiple elements prior to processing. Besides basic collection operations, a Queue provides additional insertion, extraction and inspection operations.
5. Map
   An object that maps keys to values. A map **cannot contain duplicate keys**; Each key can map to at most one value.

Collection Interface:

Important methods

Sr.No.	Method Name	Use
1	boolean add(Object ob)	Adds objects ob to the invoking collection. There are two possibilities of returning Boolean values that once if object is successfully added into collection then it returns true , it return false if object is already present in the collection. Note that collection does not have same elements twice.
2	boolean remove(object ob)	Removes a single instance of the specified element from this collection, if it is present(optional operation)and returns true Otherwise it returns false.
3	boolean addAll(Collection c)	We also add the whole collection into a new collection. It returns true if the operation is successful otherwise returns false.
4.	void clear()	Removes all elements from the collection
5.	int size()	Returns the number of elements in this collection
6.	boolean contains(Object o)	Returns true if this collection contains the specified element.
7.	boolean equals(Object o)	Compares the specified object with this collection for equality.
8.	Iterator iterator()	Returns an iterator over the elements in this collection.
9.	boolean isEmpty()	Returns true if the invoking collection is empty. Otherwise,returns false.
10.	boolean removeAll()	This is same as clear but only the difference that if

==========

		it removes the all elements it returns true otherwise it returns Flase.
11.	boolean retainAll(Collection c)	Removes all the elements from the invoking collection except those in collection c. It returns true if operation success otherwise it returns false.

iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

- public boolean hasNext() it returns true if iterator has more elements.
- public object next() it returns the element and moves the cursor pointer to the next element.
- public void remove() it removes the last elements returned by the iterator. It is rarely used.

Two ways to iterate the elements of collection in java

By Iterator interface.

By for-each loop.

Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

```
ArrayList al=new ArrayList();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

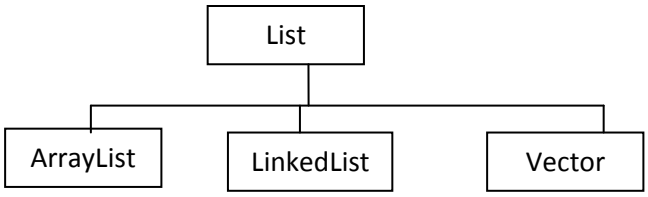
```
ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives compile time error.

List Interface

- A List cares ordered by the index
- A List is the interface which shows lists of elements.

==========



1. ArrayList (class)
   ArrayList is the best version of the array because it is available as dynamic array that grows as needed. Normally the array cannot grow or shrink as per user requirement. But collection framework gives the facility named ArrayList which gives the alternative solution for the array problem.

   Initially, ArrayList has some specific size, when the size exceeds, the collection automatically enlarged. Or when we remove element from array, the size of ArrayList automatically shrinks.

   - Ordered collection-order by index
   - The elements are **not sorted**.
   - Constructors:
     ArrayList()
     ArrayList(Collection c)
     ArrayList(int capacity)

---

interface Comparable
This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's natural natural ordering, and the class's compareTo method is referred to as its natural comparison method .
int compareTo(Object)
Returns

Negative	if object1 < object2
Zero	if object1 == object2
Positive	if object1 > object2

java.util.Collection class implements java.lang. Comparable interface
This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

==========

Provides methods for sorting

public static void sort()

Sorts the specified list into ascending order.

Iterator:

It is an interface. An object that enables you to traverse through a collection and to remove elements from the collection selectively, if desired.

Methods:

1) boolean hasNext( ) Returns true if there is a next element. Otherwise, returns false.
2) E next( ) Returns the next element. A NoSuchElementException is thrown if there is not a next element.

E specifies the type of objects being iterated.

3) E previous( ) Returns the previous element. A NoSuchElementException is thrown if there is not a previous
element. E specifies the type of objects being iterated.

4) void remove( ) Removes the current element from the list. An IllegalStateException is thrown if remove( ) is
called before next( ) or previous( ) is invoked.

2.**LinkedList** (class)

Ordered by index of the linked list is the same as ArrayList. But the difference is that elements **are doubly linked to**

- Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue. one another.
- LinkedList may iterate slowly than ArrayList.
- LinkedList is fast for insertion and deletion of items.

Constructor of the LinkedList

- LinkedList()
- LinkedList(Collection c)

ListIterator interface

An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.

==========

Methods ::

- boolean hasNext()
  Returns true if this list iterator has more elements.
- E next()
  Returns the next element in the list.
- boolean hasPrevious()
  return true if the list iterator has more elements.
- E previous()
  Returns the previous element in the list.

3.**Vector**:

- Vector implements a dynamic array same like ArrayList, but Vectors are **synchronized for thread safty**.(methods are synchronized)
- Another difference between ArrayList and Vector is that vector contains many inheritance methods that are not part of the collection framework.
- Constructors:
  - Vector()
  - Vector(int size)
  - Vector(int size,int increment)
  - Vector(Collection c)

Queue Interface In Java

The java.util.Queue is a subtype of java.util.Collection interface. It is an ordered list of objects with its use limited to inserting elements at the end of list and deleting elements from the start of list i.e. it follows FIFO principle.
Since it is an interface, we need a concrete class during its declaration. There are many ways to initialize a Queue object, most common being-

1. As a Priority Queue
2. As a LinkedList

Please note that both the implementations are not thread safe. PriorityBlockingQueue is one alternative implementation if you need a thread safe implementation.

Operations on Queue :

- **Add()-**Adds an element at the tail of queue. More specifically, at the last of linkedlist if it is used, or according to the priority in case of priority queue implementation.
- **peek()-**To view the head of queue without removing it. Returns null if queue is empty.
- **element()-**Similar to peek(). Throws NoSuchElementException if queue is empty.
- **remove()-**Removes and returns the head of the queue. Throws NoSuchElementException when queue is impty.
- **poll()-**Removes and returns the head of the queue. Returns null if queue is empty.

==========

The Set Interface

The Set interface defines a set. It extends Collection and declares the behavior of a collection **that does not allow duplicate elements**. Therefore, the add( ) method returns false if an attempt is made to add duplicate elements to a set. It does not define any additional methods of its own.

1) **HashSet**

   Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
   The important points about Java HashSet class are:

   - HashSet stores the elements by using a mechanism called **hashing.**
   - HashSet contains unique elements only.

2) **LinkedHashSet**

   It maintains a linked list of the entire in the set, in the order they are inserted. Extends HashSet to allow insertion-order iterations and adds no members of its own. It is a generic class that has this declaration:

   Class LinkedHashSet<E>

   Here, E specifies the type of objects that the set will hold. Its constructors parallel those in HashSet.

   That is, when cycling through a LinkedHashSet using an iterator, the elements will be returned in **the order in which they were inserted**.

3) **TreeSet**

   - TreeSet uses the tree for storage.
   - Objects are stored in sorted, ascending order.
   - Access and retrieval times are quite fast, which makes TreeSet an excellent choice when storing large amount of stored items done quickly.

   TreeSet has the following constructors:

   TreeSet( )
   TreeSet(Collection<? extends E>*c*)
   TreeSet(Comparator<? super E>*comp*)
   TreeSet(SortedSet<E>*ss*)

   - The first form constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements.
   - The second form builds a tree set that contains the elements of *c*.
   - The third form constructs an empty tree set that will be sorted according to the comparator specified by *comp*. (Comparators are described later in this chapter.)
   - The fourth form builds a tree set that contains the elements of *ss*.

==========

Java Map Interface

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

Map is useful if you have to search, update or delete elements on the basis of key.

Useful methods of Map interface

Method	Description
Object put(Object key, Object value)	It is used to insert an entry in this map.
void putAll(Map map)	It is used to insert the specified map in this map.
Object remove(Object key)	It is used to delete an entry for the specified key.
Object get(Object key)	It is used to return the value for the specified key.
boolean containsKey(Object key)	It is used to search the specified key from this map.
Set keySet()	It is used to return the Set view containing all the keys.
Set entrySet()	It is used to return the Set view containing all the keys and values.

Map.Entry Interface

Entry is the sub interface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key and value.

Methods of Map.Entry interface

Method	Description
Object getKey()	It is used to obtain key.
Object getValue()	It is used to obtain value.

The Map Classes

Duplicate keys are not allowed. Several classes provide implementations of the map interfaces.

1) **HashTable class**

Hashtable was part of the original java.util and is a concrete implementation of a Dictionary. Hashtable stores key/value pairs in a hash table.

Constructors:

- Hashtable( )
- Hashtable(int size)
- Hashtable(Map < ? extends K, ? extends V > t)

Methods :

- Object get(Object key)
- Object put(Object key, Object value)

2) **LinkedHashMap**

Java LinkedHashMap class is Hash table and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

The important points about Java LinkedHashMap class are:

- A LinkedHashMap contains values based on the key.
- It contains only unique elements.

==========

- It may have one null key and multiple null values.
- It is same as HashMap instead maintains insertion order.

3) HashMap

Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

A HashMap contains values based on the key.

It contains only unique elements.

It may have one null key and multiple null values.

It maintains no order.

Constructors:

Constructor	Description
HashMap()	It is used to construct a default HashMap.
HashMap(Map m)	It is used to initializes the hash map by using the elements of the given Map object m.
HashMap(int capacity)	It is used to initializes the capacity of the hash map to the given integer value, capacity.

Method	Description
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
Object put(Object key, Object value)	It is used to associate the specified value with the specified key in this map.
Collection values()	It is used to return a collection view of the values contained in this map.

4) TreeMap

- TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.
- The important points about Java TreeMap class are:
- A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- It is same as HashMap instead maintains ascending order.

Constructor	Description
TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of its key.
TreeMap(Comparator comp)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map m)	It is used to initialize a tree map with the entries from m, which will be sorted using the natural order of the keys.
TreeMap(SortedMap sm)	It is used to initialize a tree map with the entries from the SortedMap sm, which will be sorted in the same order as sm.

==========

Method	Description
boolean containsKey(Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue(Object value)	It is used to return true if this map maps one or more keys to the specified value.
Object firstKey()	It is used to return the first (lowest) key currently in this sorted map.
Object get(Object key)	It is used to return the value to which this map maps the specified key.
Object lastKey()	It is used to return the last (highest) key currently in this sorted map.
Object remove(Object key)	It is used to remove the mapping for this key from this TreeMap if present.
void putAll(Map map)	It is used to copy all of the mappings from the specified map to this map.
Set entrySet()	It is used to return a set view of the mappings contained in this map.
int size()	It is used to return the number of key-value mappings in this map.
Collection values()	It is used to return a collection view of the values contained in this map.

==========

Diffences :

HashMap	**Hashtable**
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

ArrayList	**Vector**
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

ArrayList	**LinkedList**
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.

==========

4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

	LinkedHashSet	**TreeSet**	**HashSet**
Usage	If you need to maintain order in which elements are added into Set then use LinkedHashSet, it provides ordering with little impact on performance.	Use TreeSet when you absolutely nee to keep elements in specifically sorted order e.g. keeping employees in increasing order of their age or salary.	If you need to store only unique elements without any ordering requirement.
Order	LinkedHashSet also provides ordering support to keep elements in the order they are added into Collection.	TreeSet sorts all object based upon there natural ordering by using compareTo() method, or custom order by using compare() method Comparator passed to them.	Not guaranteed
Data structure	LinkedHashSet internally uses LinkedHashMap	TreeSet internally uses TreeMap. Red- black Tree	HashSet internally uses HashMap
Peroformance	LinkedHashSet is close second, as it suffers a little drop in performance due to overhead of maintaining doubly linked list when an element is inserted or deleted.	TreeSet is much slower than these two because it needs to perform sorting every time there is change in TreeSet.	HashSet is fastest for common operation e.g. add, search and remove

	HashSet	**HashMap**
Implementation	HashSet class implements the Set interface	HashMap class implements the Map interface
Example	HashSet, we store objects(elements or values) e.g. If we have a HashSet of string elements then it could depict a set of HashSet elements: {“Hello”, “Hi”, “Bye”, “Run”}	HashMap is used for storing key & value pairs. In short, it maintains the mapping of key & value (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This is how you could represent HashMap elements if it has integer key and value of String type: e.g. {1->”Hello”, 2->”Hi”, 3->”Bye”, 4->”Run”}
Duplicate	HashSet does not allow duplicate elements that mean you can not store duplicate values in HashSet.	HashMap does not allow duplicate keys however it allows having duplicate values.
Null	HashSet permits to have a single null value.	HashMap permits single null key and any number of null values.
Similarity	HashSet is not synchronized which means they are not suitable for thread-safe operations until unless synchronized explicitly.	HashMap is not synchronized which means they are not suitable for thread-safe operations until unless synchronized explicitly.

==========

Java Regex

The Java Regex or Regular Expression is an API to *define pattern for searching or manipulating strings*.

It is widely used to define constraint on strings such as password and email validation. Java Regex API provides 1 interface and 3 classes in java.util.regex package.

java.util.regex package

It provides following classes and interface for regular expressions. The Matcher and Pattern classes are widely used in java regular expression.

MatchResult interface

Matcher class

Pattern class

PatternSyntaxException class

Matcher class

It implements MatchResult interface. It is a *regex engine* i.e. used to perform match operations on a character sequence.

No.	Method	Description
1	boolean matches()	test whether the regular expression matches the pattern.
2	boolean find()	finds the next expression that matches the pattern.
3	boolean find(int start)	finds the next expression that matches the pattern from the given start number.
4	String group()	returns the matched subsequence.
5	int start()	returns the starting index of the matched subsequence.
6	int end()	returns the ending index of the matched subsequence.
7	int groupCount()	returns the total number of the matched subsequence.

Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

No.	Method	Description
1	static Pattern compile(String regex)	compiles the given regex and return the instance of pattern.
2	Matcher matcher(CharSequence input)	creates a matcher that matches the given input with pattern.
3	static boolean matches(String regex, CharSequence input)	It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern.
4	String[] split(CharSequence input)	splits the given input string around matches of given pattern.
5	String pattern()	returns the regex pattern.

==========

Regex Character classes

No.	Character Class	Description
1	[abc]	a, b, or c (simple class)
2	[^abc]	Any character except a, b, or c (negation)
3	[a-zA-Z]	a through z or A through Z, inclusive (range)
4	[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
5	[a-z&&[def]]	d, e, or f (intersection)
6	[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
7	[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Regex Quantifiers

The quantifiers specify the number of occurrences of a character.

Regex	Description
X?	X occurs once or not at all
X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

==========

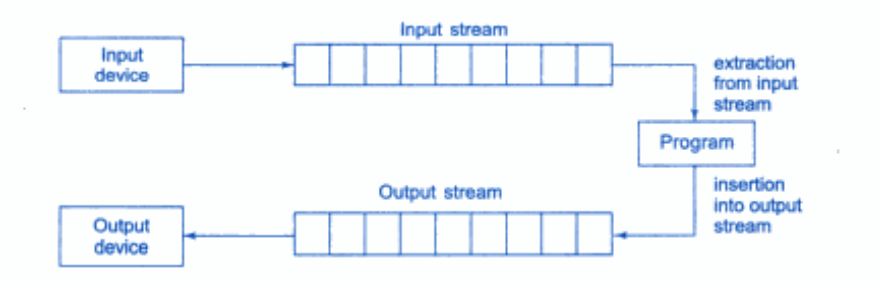
FILE I/O

Streams

Java programs perform I/O through streams. A *stream* is an abstraction that either produces or consumes information. A stream is linked to a physical device by the Java I/O system.

The stream is a sequence of bytes. It acts as a source from which from the input data can be obtained or a destination to which the output data can be sent.

A program extracts data from input stream and inserts the data to output stream.



File class

That is, the **File** class **does not** specify how information is **retrieved from or stored in files**; it describes the properties of a file itself.

Constructors:

- File(String *directoryPath*)
- File(String *directoryPath*, String *filename*)
- File(File *dirObj*, String *filename*)
- File(URI *uriObj*)

Methods:

- public String getName()
  Returns the name of the file or directory denoted by this abstract pathname.
- public String getParent()
  Returns the pathname string of this abstract pathname's parent
- File getParentFile()
  Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
- boolean exists()
  Tests whether the file or directory denoted by this abstract pathname exists.
- String getPath()
  Converts this abstract pathname into a pathname string.
- boolean isAbsolute()

==========

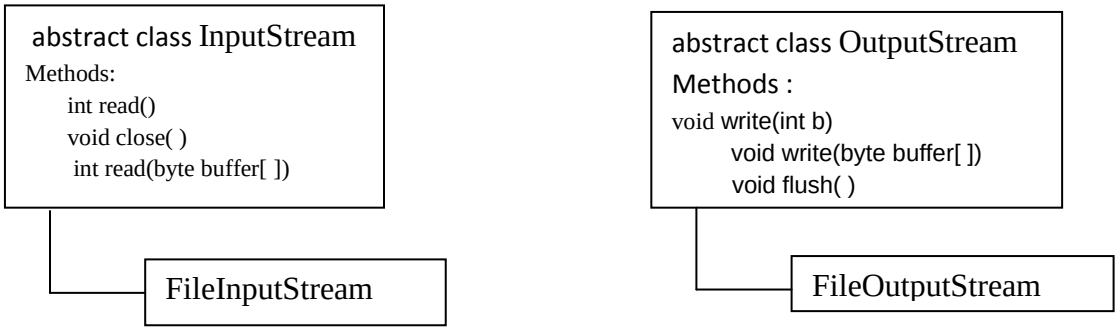
Input Stream:
disk file,a keyboard, or a network socket

Output Stream
a disk file, or a network connection

Byte Streams and Character Streams

Byte streams:
Provide a convenient means for handling input and output of bytes. Byte streams are used for reading or writing binary data.
Character streams:
Provide a convenient means for handling input and output of characters. They use Unicode and, therefore, can be internationalized.

The Byte Stream Classes



FileInputStream:
The FileInputStream class creates an InputStream that you can use to read bytes from a file.
Constructors:

- FileInputStream(String *filepath*) //Full path of the file
- FileInputStream(File *fileObj*)

Methods throw a FileNotFoundException

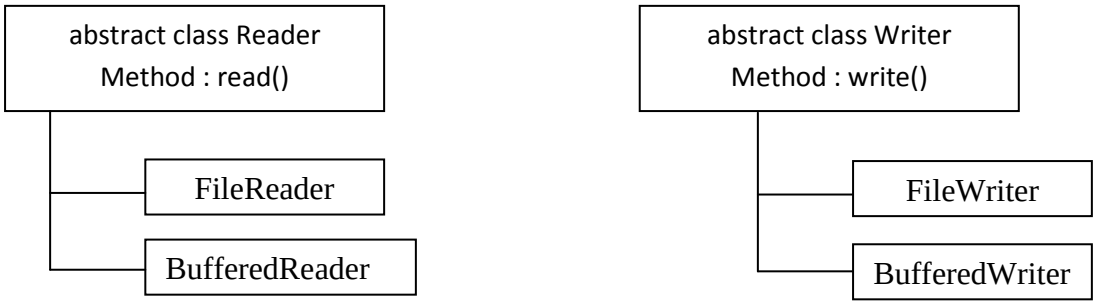
FileOutputStream:
FileOutputStream creates an OutputStream that you can use to write bytes to a file.
Constructors:

- FileOutputStream(String *filePath*) //full path
- FileOutputStream(File *fileObj*)
- FileOutputStream(String *filePath*, boolean *append*)
- FileOutputStream(File *fileObj*, boolean *append*)

Methods throw a FileNotFoundException

==========

The Character Stream Classes



FileReader
The **FileReader** class creates a **Reader** that you can use to read the contents of a file.
Constructors:

- FileReader(String *filePath*)
- FileReader(File *fileObj*)

FileWriter
FileWriter creates a Writer that you can use to write to a file.

Constructors:

- FileWriter(String *filePath*)
- FileWriter(String *filePath*, boolean *append*)
- FileWriter(File *fileObj*)
- FileWriter(File *fileObj*, boolean *append*)

The Predefined Streams
System.outrefers to the standard output stream. By default, this is the console. System**.**in refers to standard input, which is the keyboard by default. System**.**errrefers to the standard error stream, which also is the console by default.
System**.**outis a byte stream, using it for simple program output is still acceptable.

RandomAccessFile
RandomAccessFile encapsulates a random-access file. It is not derived from InputStream or OutputStream.
RandomAccessFile is special because it supports positioning requests—that is, you can position the file pointerwithin the file.
File access:

- "r" = file is opened in read mode
- "rw" = file is opened in read-write mode

==========

Constructors:

RandomAccessFile(File *fileObj*, String *access*) throws FileNotFoundException
RandomAccessFile(String *filename*, String *access*) throws FileNotFoundException

Methods

- void seek(long *newPos*) throws IOException
  is used to set the current position of the file pointer within the file.
- long length() returns number of character of the file(SIZE)
- void setLength(long *len*) throws IOException
  This method sets the length of the invoking file to that specified by *len.* This method can be used to lengthen or shorten a file.

Serialization

Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

Serializable

Only an object that implements the Serializableinterface can be saved and restored by the serialization facilities. The Serializableinterface defines no members. It is simply used to indicate that a class may be serialized. If a class is serializable, all of its subclasses are also serializable.

Variables that are declared as **transient** are not saved by the serialization facilities. Also, **static** variables are not saved.

Note: PrintStream is an output stream derived from OutputStream**.**

==========

==========

Applet

Applet is a java program which runs on browser.It is client side program that can be included in HTML page.To run applet we need browser.

Applet class is contained in java.applet package

Execution of applet does not begin at (main()) method ,but few applet may have main() method.
HTML tag:-

To execute applet on broweser you need to write short HTML page that contains the appropriate HTML tags.

```
/*
<applet code="ClassName" width=”300” height="300">
        </applet>
*/
```

Points to remember:-
1) Applet do not have main() method.
2) The Applet class should be declared as public.
3) Applet must be run under an appletviewer or for java compatible browser.

Applet Life Cycle:

1) init()

init() is the first method to be called.This method is called only once during the life time of the Applet.

2) start()

This method is called by init() method. It is called to restart aooket cycke which has been stoped.

So if user use a web page and comes back to the applet then execution became start from start() method.

3) paint()

The paint() method is called each time your applet output must be redrawn. paint method have parameter of Graphics object.

```
public void paint(Graphics g)
 {  }
```

The methods of Graphics class are use to draw out put.Graphics class is in awt package

e.g. void drawString(String, int, int)
void drawLine(int,int,int,int)

4) stop()

It is called when a web browser use the web page document containing applet. You should respect thread do not run when the applet is visible.

==========

5) destroy()

destroy() method is called when the envioronment determines that your applet needs to be removed completely from memory. The stop() method is always called before destroy() method.

Praam tag:

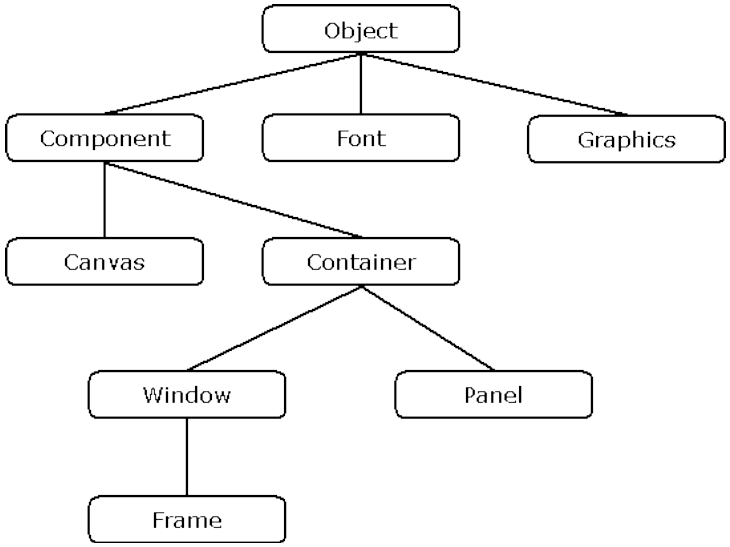
param tag is used to send parameters from HTML page to applet. It is used with applet tag(Sub tag of applet)

```
/*
    <applet code="ClassName" width=300 height="300">

        <param name="n" value="Hello" />

    </applet>
*/
```

Important methods of the Graphics class

1)public abstract void setColor(java.awt.Color);
2)public abstract void setFont(java.awt.Font);
3)public abstract void drawOval(int, int, int, int);
4)public abstract void fillOval(int, int, int, int);
5)public abstract void drawArc(int, int, int, int, int, int);
6)public abstract void fillArc(int, int, int, int, int, int);
7)public abstract void drawString(java.lang.String, int, int);
8) public void drawBytes(byte[], int, int, int, int);
9)public abstract boolean drawImage(java.awt.Image, int, int,
java.awt.image.ImageObserver);
10) public abstract void dispose();

==========



Important mehods of the Component class

1) public void setBackground(java.awt.Color);
2) public void setForeground(java.awt.Color);
3)public void setVisible(boolean);
4)public void setBounds(int, int, int, int);
5)public int getX();
6) public int getY();
7) public void paint(java.awt.Graphics);
8)public void update(java.awt.Graphics);

Note: Whenever yor applet needs to update the information displayed in its window,it simply calls repaint().This causes the AWTrun-time system to execute a call to your applet's update() method which ,in its default implentation,calls paint().

Forms of repaint() method of AWT

1.void repaint()
2.void repaint(int left,int top,int width,int height)
3.void void repaint(long maxDelay)
4.void repaint(long maxDelay,int x,int y,int width,int height)

Important mehods of the Container class

1)public void add(java.awt.Component);
2)void add(Component obj,int index)
3)public void remove(int);
4)public void remove(java.awt.Component);
5)public void removeAll();

==========

6)public void setLayout(java.awt.LayoutManager);
7)public void setFont(java.awt.Font);

Event Hadling:

When you press Button,selct value from List or ComboBox or Checkbox or radio , move scroll bar, press.relese,click mouse the object of Event Handling classes are passed to particular methods of interface.

1) ActionEvent :

Generated when button is pressed,list item is double clicked and menu item is selected.

Methods:

String getActionCommand()
int getModifier()

Constants :-

ACTION_FIRST
ACTION_LAST
ACTION_PERFORMED

2) ItemEvent:

Generated when checkbox or list are clicked also occures when a choice selection is made.

Methods:

int getStateChanged()
Object getItem()

Constants :-

ITEM_FIRST
ITEM_LAST
ITEM_STATE_CHANGED
SELECTED
DESELECTED

3) AdjustmentEvent:

Generated when scroll bar is amnipulated.

Methods:

int getValue()
int getAdjustmentType()

Constants :-

==========

ADJUSTMENT_FIRST
- ADJUSTMENT_LAST
- ADJUSTMENT_VALUE_CHANGED

4) MouseEvent:

Generated when mouse dragged,moved,clicked,pressed or relesed.Also generated when mouse is entered or exited from component.

Methods:

int getX()
int getY()
inr getClickCount()

Constants :-

MOUSE_CLICKED
MOUSE_PRESSED
MOUSE_MOVED
MOUSE_ENTERED
MOUSE_EXITED
MOUSE_DRAGGED

5) TextEvent:

Generated when value of text field is changed

6) KeyEvent

Generaed when key is pressed or relesed.

Methods:

char getKeyChar()

Constants :-

KEY_PRESSED
KEY_TYPED
KEY_RELEASED

7) WidndowEvent

Generated when widow is closed,closing and opened.

Methods:

Widndow getWindow()

Constants :-

WINDOW_OPENED

==========

WINDOW_CLOSING
WINDOW_CLOSED
WINDOW_ICONIFIED
WINDOW_DEICONIFIED

8) FocusEvent

Generated when focus gained or lost.

Methods:

boolean isTemporary()

Constants :-

FOCUS_FIRST
FOCUS_LAST
FOCUS_GAINED
FOCUS_LOST

Event Listener (Interfaces):

A listener is an object that indicates it wants to receive events of particular type.

ActionListener

public void actionPerformed(ActionEvent ae)

ItemListener

public void itemStateChanged(ItemEvent ie)

MouseLIstener

public void mouseClicked(MouseEvent ie)
public void mousePressed(MouseEvent ie)
public void mouseReleased(MouseEvent ie)
public void mouseEntered(MouseEvent ie)
public void mouseExited(MouseEvent ie)

MouseMotionListener

public void mouseDragged(MouseEvent ie)
public void mouseMoved(MouseEvent ie)

AdjustmentListener

public void adjustmentValueChanged(AdjustmentEvent ae)

KeyListener

public void keyTyped(KeyEvent ke)
public void keyPressed(KeyEvent ke)
public void keyReleased(KeyEvent ke)

==========

WindowListener
public void widowActivated(WindowEvent we)
public void widowDeactivated(WindowEvent we)
public void widowClosing(WindowEvent we)
public void widowClosed(WindowEvent we)
public void widowOpened(WindowEvent we)
public void widowIconified(WindowEvent we)
public void widowDeiconified(WindowEvent we)

AWT component that fiers event:
Frame,Diaolog

FocusListener
void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)

==========
public class Font:
Font is used to set fonts while paint on graphics .You can set font family,font style and font size.
Constructors:
Font()
Font(String fontfamily,int fontStyle, int size)

Methods:
Font setName()
int getSize()
int getStyle()
String getFamily()

Constant:
public static final java.lang.String DIALOG;
public static final java.lang.String DIALOG_INPUT;
public static final java.lang.String SANS_SERIF;
public static final java.lang.String SERIF;
public static final java.lang.String MONOSPACED;
public static final int PLAIN; // index 0
public static final int BOLD; // index 1
public static final int ITALIC; // index 2

public static final int ROMAN_BASELINE;
public static final int CENTER_BASELINE;

==========

setFont() Method of Graphics class or

Method of Container is used to set font.
void setFont(Font obj)

==========

java.awt.Color class
Constructor:
Color(int, int, int);
Color(float, float, float);

Methods:
public int getRed();
public int getGreen();
public int getBlue();

Static type Constants :
Color white;
Color WHITE;
Color lightGray;
Color LIGHT_GRAY;
Color gray; Color GRAY Color darkGray Color DARK_GRAY
Color black Color BLACK; Color red; Color RED; Color pink;
Color PINK; Color orange; Color ORANGE; Color yellow; Color YELLOW;
Color green; Color GREEN; Color.magenta; Color MAGENTA; Color cyan;
Color CYAN; Color blue; Color BLUE;

Methods component class used to set color for backgroung and forground:
void setBackground(Color obj)
void setForeground(Color obj)
void setBackground(int ColorConstant)
void setForeground(int ColorConstant)

Method of Graphics class Used to set color
void setColor(Color obj)

---

class - public java.awt.Label

Construcors
Label()
Label(java.lang.String)
Label(String, int)

Methods:-

==========

void setText(String)
String getText()
void setAlignment(int how)
here how means pass constant for left,center,right
String getAlignment()

Constants:
public static final int LEFT
public static final int CENTER
public static final int RIGHT

---

java.awt.TextField
Constuctors:
TextField()
TextField(java.lang.String)
TextField(int)
public java.awt.TextField(java.lang.String, int)

Important Methods:
public char getEchoChar();
public void setEchoChar(char);
public void setText(java.lang.String);
public void setText(java.lang.String);
String getSelectedText()

==========

java.awt.Button
Constuctors:
public java.awt.Button() throws ja
public java.awt.Button(java.lang.String)

Important Methods
String getLabel()
void setLabel(java.lang.String)

==========

java.awt.TextArea

Constuctors:
Important Methods:
void insert(String, int)
void insertText(String, int)
void append(String)
void appendText(String)
void replaceRange(String, int, int)

==========

Important Methods:
void insert(String, int)
void insertText(String, int)
void append(String)
void appendText(String)
void replaceRange(String, int, int)

Constants:
public static final int SCROLLBARS_BOTH
public static final int SCROLLBARS_VERTICAL_ONLY
public static final int SCROLLBARS_HORIZONTAL_ONLY
public static final int SCROLLBARS_NONE

==========

class java.awt.Checkbox
Constuctors:
Checkbox()
Checkbox(java.lang.String)
Checkbox(java.lang.String, boolean)
Checkbox(java.lang.String, boolean, java.awt.CheckboxGroup)
Checkbox(java.lang.String, java.awt.CheckboxGroup, boolean)

Important Methods:
String getLabel();
public void setLabel(java.lang.String);
public boolean getState();
public void setState(boolean);

---

java.awt.CheckboxGroup
Constuctors:
CheckboxGroup();

Important Methods:
Checkbox getSelectedCheckbox();
Checkbox getCurrent();
void setSelectedCheckbox(java.awt.Checkbox);
void setCurrent(java.awt.Checkbox);

---

java.awt.List

==========

Constuctors:
public java.awt.List()
public java.awt.List(int)
public java.awt.List(int, boolean)

Important Methods:
String getSelectedItem()
String[] getSelectedItems()
String getItem(int)
int getItemCount();
int countItems()
int getSelectedIndex()
int[] getSelectedIndexes()
void select(int)
boolean isIndexSelected(int);
boolean isSelected(int)
void makeVisible(int)
void remove(int)
void removeAll()
void remove(String)

---

java.awt.Choice
Constuctors:
Choice()
Important Methods:
int getItemCount()
int countItems()
String getItem(int)
String getItemImpl(int)
void add(java.lang.String)
void addItem(java.lang.String)
void insert(java.lang.String, int)
void remove(java.lang.String)
void remove(int)
void removeAll()
String getSelectedItem()
int getSelectedIndex()
void select(int)
void select(String)

---

class java.awt.Menu
Constuctors:
Menu()

==========

Menu(java.lang.String)
Menu(.String, boolean)

Important Methods:
int getItemCount()
int countItems()
MenuItem getItem(int)
MenuItem add(java.awt.MenuItem)
void add(java.lang.String)
void insert(java.awt.MenuItem, int)
void insert(java.lang.String, int)
void remove(int);
void remove(java.awt.MenuComponent);
void removeAll();

---

java.awt.MenuItem

Constuctors:
MenuItem()
MenuItem(java.lang.String)
MenuItem(java.lang.String, java.awt.MenuShortcut)

Important Methods:
String getLabel()
void setLabel(java.lang.String)
public boolean isEnabled();
void setEnabled(boolean)

---

java.awt.MenuShortcut

Constuctors:
MenuShortcut(int);
MenuShortcut(int, boolean);

Important Methods:
public int getKey();

public boolean equals(java.awt.MenuShortcut);
public boolean equals(java.lang.Object);

---

class java.awt.Graphics

1)public abstract void setColor(java.awt.Color);
2)public abstract void setFont(java.awt.Font);

==========

3)public abstract void drawOval(int, int, int, int);
4)public abstract void fillOval(int, int, int, int);
5)public abstract void drawArc(int, int, int, int, int, int);
6)public abstract void fillArc(int, int, int, int, int, int);
7)public abstract void drawString(java.lang.String, int, int);
8) public void drawBytes(byte[], int, int, int, int);
9)public abstract boolean drawImage(java.awt.Image, int, int,
java.awt.image.ImageObserver);
10) public abstract void dispose();

==========

Java Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
Unlike AWT, Java Swing provides platform-independent and lightweight components.

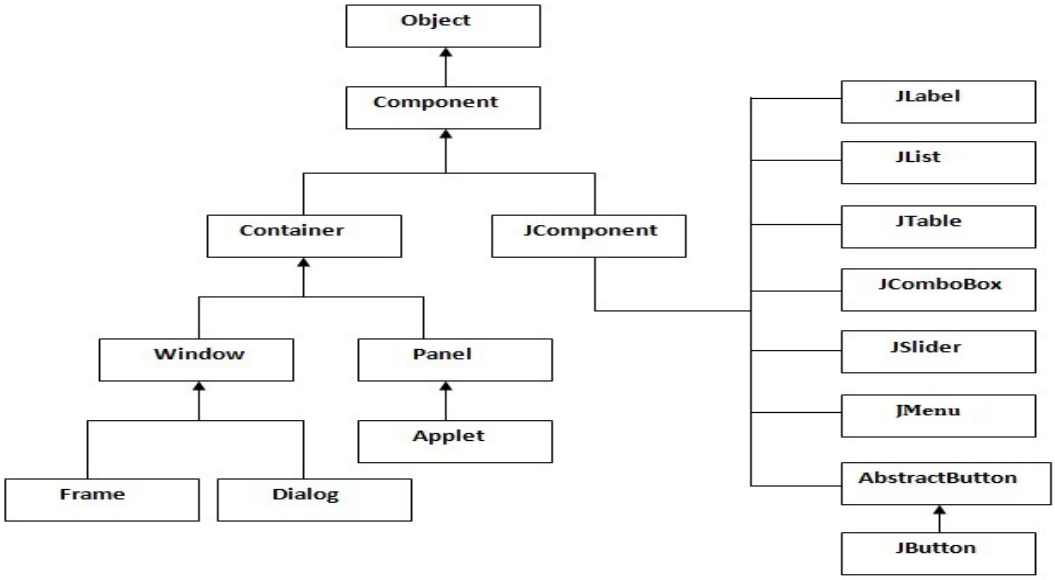
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing
There are many differences between java awt and swing that are given below.

.	**Java AWT**	**Java Swing**
1)	AWT components are platform-dependent.	Java swing components are platform-independent.
2)	AWT components are heavyweight.	Swing components are lightweight.
3)	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

What is JFC
The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
Hierarchy of Java Swing classes

==========



Commonly used Methods of Component class

Method	**Description**
public void add(Component c)	Add A Component On Another Component.
public void setSize(int width,int height)	Sets Size Of The Component.
public void setLayout(LayoutManager m)	Sets The Layout Manager For The Component.
public void setVisible(boolean b)	Sets The Visibility Of The Component. It Is By Default False.

---

1) JLabel

2) Class constructors

S.N.	Constructor & Description
1	JLabel() Creates a JLabel instance with no image and with an empty string for the title.
2	JLabel(Icon image) Creates a JLabel instance with the specified image.
3	JLabel(Icon image, int horizontalAlignment) Creates a JLabel instance with the specified image and horizontal alignment.
4	JLabel(String text) Creates a JLabel instance with the specified text.
5	JLabel(String text, Icon icon, int horizontalAlignment) Creates a JLabel instance with the specified text, image, and horizontal alignment.
6	JLabel(String text, int horizontalAlignment) Creates a JLabel instance with the specified text and horizontal alignment.

3) Class methods

==========

S.N.	Method & Description
1	protected int checkHorizontalKey(int key, String message) Verify that key is a legal value for the horizontalAlignment properties.
2	protected int checkVerticalKey(int key, String message) Verify that key is a legal value for the verticalAlignment or verticalTextPosition properties.
3	AccessibleContext getAccessibleContext() Get the AccessibleContext of this object.
4	Icon getDisabledIcon() Returns the icon used by the label when it's disabled.
5	int getDisplayedMnemonic() Return the keycode that indicates a mnemonic key.
6	int getDisplayedMnemonicIndex() Returns the character, as an index, that the look and feel should provide decoration for as representing the mnemonic character.
7	int getHorizontalAlignment() Returns the alignment of the label's contents along the X axis.
8	int getHorizontalTextPosition() Returns the horizontal position of the label's text, relative to its image.
9	Icon getIcon() Returns the graphic image (glyph, icon) that the label displays.
10	int getIconTextGap() Returns the amount of space between the text and the icon displayed in this label.
11	Component getLabelFor() Get the component this is labelling.
12	String getText() Returns the text string that the label displays.
13	LabelUI getUI() Returns the L&F object that renders this component.
14	String getUIClassID() Returns a string that specifies the name of the l&f class that renders this component.
15	int getVerticalAlignment() Returns the alignment of the label's contents along the Y axis.
16	int getVerticalTextPosition() Returns the vertical position of the label's text, relative to its image.
17	boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h) This is overridden to return false if the current Icon's Image is not equal to the passed in Image img.
18	protected String paramString() Returns a string representation of this JLabel.
19	void setDisabledIcon(Icon disabledIcon) Set the icon to be displayed if this JLabel is "disabled" (JLabel.setEnabled(false)).
20	void setDisplayedMnemonic(char aChar) Specifies the displayedMnemonic as a char value.
21	void setDisplayedMnemonic(int key) Specify a keycode that indicates a mnemonic key.
22	void setDisplayedMnemonicIndex(int index) Provides a hint to the look and feel as to which character in the text should be decorated to represent the mnemonic.
23	void setHorizontalAlignment(int alignment) Sets the alignment of the label's contents along the X axis.
24	void setHorizontalTextPosition(int textPosition)

==========

	Sets the horizontal position of the label's text, relative to its image.
25	void setIcon(Icon icon) Defines the icon this component will display.
26	void setIconTextGap(int iconTextGap) If both the icon and text properties are set, this property defines the space between them.
27	void setLabelFor(Component c) Set the component this is labelling.
28	void setText(String text) Defines the single line of text this component will display.
29	void setUI(LabelUI ui) Sets the L&F object that renders this component.
30	void setVerticalAlignment(int alignment) Sets the alignment of the label's contents along the Y axis.
31	void setVerticalTextPosition(int textPosition) Sets the vertical position of the label's text, relative to its image.
32	void updateUI() Resets the UI property to a value from the current look and feel.

4) JButton class:

The JButton class is used to create a button that have plateform-independent implementation.

Commonly used Constructors:

- JButton(): creates a button with no text and icon.
- JButton(String s): creates a button with the specified text.
- JButton(Icon i): creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

1) public void setText(String s): is used to set specified text on button.
2) public String getText(): is used to return the text of the button.
3) public void setEnabled(boolean b): is used to enable or disable the button.
4) public void setIcon(Icon b): is used to set the specified Icon on the button.
5) public Icon getIcon(): is used to get the Icon of the button.
6) public void setMnemonic(int a): is used to set the mnemonic on the button.
7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

---

5) ButtonGroup Class

How to Use the ButtonGroup Component

The ButtonGroup component manages the selected/unselected state for a set of buttons. For the group, the ButtonGroup instance guarantees that only one button can be selected at a time. Initially, all buttons managed by a ButtonGroup instance are unselected.

==========

How to Use ButtonGroup Features

You can use ButtonGroup with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton, JRadioButtonMenuItem, or JToggleButton. It would not make sense to put an instance of JButton or JMenuItem in a button group because JButton and JMenuItem do not implement the select/deselect button state.

In general, you will typically follow these steps to write code that uses a ButtonGroup component.

a) Subclass JFrame
b) Call ContextPane together with a layout manager
c) Declare and configure a set of radio buttons or toggle buttons
d) Instantiate a ButtonGroup object
e) Call the add method on that buttongroup object in order to add each button to the group.

Commonly Used Button Group Constructors/Methods	
Constructor or Method	**Purpose**
ButtonGroup()	Create a ButtonGroup instance.
void add(AbstractButton) void remove(AbstractButton)	Add a button to the group, or remove a button from the group.
public ButtonGroup getGroup() *(in DefaultButtonModel)*	Get the ButtonGroup, if any, that controls a button. For example: ButtonGroup group = ((DefaultButtonModel)button.getModel()).getGroup();
public ButtonGroup clearSelection()	Clears the state of selected buttons in the ButtonGroup. None of the buttons in the ButtonGroup are selected .

---

6) JCheckbox

Constructors

JCheckBox()

A. JCheckBox(java.lang.String text)
B.
C. JCheckBox(java.lang.String text, boolean selected)

D. JCheckBox(java.lang.String text, boolean selected, ButtonGroup bg)
E. This constructor is used for AWT CheckboxGroup support
F. JCheckBox(java.lang.String text, Icon icon)

Method Summary

a) protected Dimension calculatePreferredSize()

==========

b) Overriden to calculate non-realised preferred size.
c) int getMnemonic()
d) java.lang.Object[] getSelectedObjects()
e) boolean isArmed()
f) boolean isPressed()
g) boolean isRollover()
h) boolean isSelected()
i) void processKeyEvent(KeyEvent e)
j) Sends KeyEvents to component listeners Overriden from Component to handle ItemEvents
k) void processMouseEvent(MouseEvent e)
l) Sends mouse events to component listeners Overriden from Component to handle ItemEvents
m) void setArmed(boolean b)
n) void setPressed(boolean b)
o) void setRollover(boolean b)
p) void setSelected(boolean b)
q) void setSwingWTParent(Container parent)
r) Once a parent component receives an "add" call for a child, this being the child, this should be called to tell us to instantiate the peer and load in any cached properties.

---

7) JRadioButton class

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
It should be added in ButtonGroup to select one radio button only.

Commonly used Constructors of JRadioButton class:

- JRadioButton(): creates an unselected radio button with no text.
- JRadioButton(String s): creates an unselected radio button with specified text.
- JRadioButton(String s, boolean selected): creates a radio button with the specified text and selected status.

Commonly used Methods of AbstractButton class:

1) public void setText(String s): is used to set specified text on button.
2) public String getText(): is used to return the text of the button.
3) public void setEnabled(boolean b): is used to enable or disable the button.
4) public void setIcon(Icon b): is used to set the specified Icon on the button.
5) public Icon getIcon(): is used to get the Icon of the button.
6) public void setMnemonic(int a): is used to set the mnemonic on the button.

==========

7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

---

8) JList

JList constructors

The following constructors create a JList instances:

public JList() -- creates an instance of a JList object with an empty data model.

public JList(ListModel dataModel) -- creates an instance of a JList object with the data model specified in the supplied parameter. The data model must not be empty.

public JList(Object listData[]) -- creates an instance of a JList object with the data specified in the supplied array. This array is inserted into a default list model.

public JList(Vector listData) -- creates an instance of a JList object with the data specified in the supplied vector. This array is inserted into a default list model.

---

9) JTextField Class

The class JTextField is a component which allows the editing of a single line of text.

Class constructors

S.N.	Constructor & Description
1	JTextField() Constructs a new TextField.
2	JTextField(Document doc, String text, int columns) Constructs a new JTextField that uses the given text storage model and the given number of columns.
3	JTextField(int columns) Constructs a new empty TextField with the specified number of columns.
4	JTextField(String text) Constructs a new TextField initialized with the specified text.
5	JTextField(String text, int columns) Constructs a new TextField initialized with the specified text and columns.

Class methods

S.N.	Method & Description
1	protected void actionPropertyChanged(Action action, String propertyName) Updates the textfield's state in response to property changes in associated action.
2	void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this textfield.
3	protected void configurePropertiesFromAction(Action a) Sets the properties on this textfield to match those in the specified Action.
4	protected PropertyChangeListener createActionPropertyChangeListener(Action a) Creates and returns a PropertyChangeListener that is responsible for listening for changes from the specified Action and updating the appropriate properties.

==========

5	protected Document createDefaultModel() Creates the default implementation of the model to be used at construction if one isn't explicitly given.
6	protected void fireActionPerformed() Notifies all listeners that have registered interest for notification on this event type.
7	AccessibleContext getAccessibleContext() Gets the AccessibleContext associated with this JTextField.
8	Action getAction() Returns the currently set Action for this ActionEvent source, or null if no Action is set.
9	ActionListener[] getActionListeners() Returns an array of all the ActionListeners added to this JTextField with addActionListener().
10	Action[] getActions() Fetches the command list for the editor.
11	int getColumns() Returns the number of columns in this TextField.
12	protected int getColumnWidth() Returns the column width.
13	int getHorizontalAlignment() Returns the horizontal alignment of the text.
14	BoundedRangeModel getHorizontalVisibility() Gets the visibility of the text field.
15	Dimension getPreferredSize() Returns the preferred size Dimensions needed for this TextField.
16	int getScrollOffset() Gets the scroll offset, in pixels.
17	String getUIClassID() Gets the class ID for a UI.
18	boolean isValidateRoot() Calls to revalidate that come from within the textfield itself will be handled by validating the textfield, unless the textfield is contained within a JViewport, in which case this returns false.
19	protected String paramString() Returns a string representation of this JTextField.
20	void postActionEvent() Processes action events occurring on this textfield by dispatching them to any registered ActionListener objects.
21	void removeActionListener(ActionListener l) Removes the specified action listener so that it no longer receives action events from this textfield.
22	void scrollRectToVisible(Rectangle r) Scrolls the field left or right.
23	void setAction(Action a) Sets the Action for the ActionEvent source.

==========

24	void setActionCommand(String command) Sets the command string used for action events.
25	void setColumns(int columns) Sets the number of columns in this TextField, and then invalidate the layout.
26	void setDocument(Document doc) Associates the editor with a text document.
27	void setFont(Font f) Sets the current font.
28	void setHorizontalAlignment(int alignment) Sets the horizontal alignment of the text.
29	void setScrollOffset(int scrollOffset) Sets the scroll offset, in pixels.

---

10) JTextArea class

The JTextArea class is used to create a text area. It is a multiline area that displays the plain text only.

Commonly used Constructors:

- JTextArea(): creates a text area that displays no text initially.
- JTextArea(String s): creates a text area that displays specified text initially.
- JTextArea(int row, int column): creates a text area with the specified number of rows and columns that displays no text initially..
- JTextArea(String s, int row, int column): creates a text area with the specified number of rows and columns that displays specified text.

Commonly used methods of JTextArea class:

1) public void setRows(int rows): is used to set specified number of rows.
2) public void setColumns(int cols):: is used to set specified number of columns.
3) public void setFont(Font f): is used to set the specified font.
4) public void insert(String s, int position): is used to insert the specified text on the specified position.
5) public void append(String s): is used to append the given text to the end of the document.

---

11) JComboBox class:

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

Commonly used Constructors of JComboBox class:

- JComboBox()
- JComboBox(Object[] items)

==========

- JComboBox(Vector<?> items)

Commonly used methods of JComboBox class:

1) public void addItem(Object anObject): is used to add an item to the item list.
2) public void removeItem(Object anObject): is used to delete an item to the item list.
3) public void removeAllItems(): is used to remove all the items from the list.
4) public void setEditable(boolean b): is used to determine whether the JComboBox is editable.
5) public void addActionListener(ActionListener a): is used to add the ActionListener.
6) public void addItemListener(ItemListener i): is used to add the ItemListener.

JScrollPane Class

Package: javax.swing

The JScrollPane class creates scroll bars, which you use in conjunction with other controls such as text areas or list controls.

Note: This section doesn't list any methods for the JScrollPane class. The JScrollPane class does have methods (plenty of them, in fact), but none of them are particularly useful for ordinary programming.

Constructors

Constructor	Description
JScrollPane(Component view)	Creates a scroll pane for the specified component
JScrollPane(Component, int vert, int hor)	Creates a scroll pane for the specified component with the specified policy for the vertical and horizontal scroll bars

Fields

Field	Description
VERTICAL_SCROLLBAR_ALWAYS	Always adds a vertical scroll bar
VERTICAL_SCROLLBAR_AS_NEEDED	Adds a vertical scroll bar if necessary
VERTICAL_SCROLLBAR_NEVER	Never adds a vertical scroll bar
HORIZONTAL_SCROLLBAR_ALWAYS	Always adds a horizontal scroll bar
HORIZONTAL_SCROLLBAR_AS_NEEDED	Adds a horizontal scroll bar if necessary
HORIZONTAL_SCROLLBAR_NEVER	Never adds a horizontal scroll bar

---

12) JPanel class

Constructors

a) JPanel()
b) Creates a new JPanel with a double buffer and a flow layout.

==========

c) JPanel(boolean isDoubleBuffered)
d) Creates a new JPanel with FlowLayout and the specified buffering strategy.
e) JPanel(LayoutManager layout)
f) Create a new buffered JPanel with the specified layout manager
g) JPanel(LayoutManager layout, boolean isDoubleBuffered)
h) Creates a new JPanel with the specified layout manager and buffering strategy.

Method

a) updateUI

public void updateUI()

Resets the UI property with a value from the current look and feel.

Overrides: updateUI in class JComponent

b) getUI

public PanelUI getUI()

Returns the look and feel (L&F) object that renders this component.

Returns:

the PanelUI object that renders this component

c) setUI

public void setUI(PanelUI ui)

Sets the look and feel (L&F) object that renders this component.

Parameters:

ui - the PanelUI L&F object

Since:

d) getUIClassID

public String getUIClassID()

Returns a string that specifies the name of the L&F class that renders this component.

Overrides:

getUIClassID in class JComponent

Returns:

"PanelUI"

e) paramString

protected String paramString()

Returns a string representation of this JPanel. This method is intended to be used only for debugging purposes, and the content and format of the returned string may vary between implementations. The returned string may be empty but may not be null.

Returns:

A string representation of this JPanel.

f) getAccessibleContext

==========

public AccessibleContext getAccessibleContext()
Gets the AccessibleContext associated with this JPanel. For JPanels, the AccessibleContext takes the form of an AccessibleJPanel. A new AccessibleJPanel instance is created if necessary.
Returns:
an AccessibleJPanel that serves as the AccessibleContext of this JPanel

---

13) JTable class

The JTable class is used to display the data on two dimensional tables of cells.
Commonly used Constructors of JTable class:

JTable(): creates a table with empty cells.
JTable(Object[][] rows, Object[] columns): creates a table with the specified data.

---

14) JColorChooser class:

The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

Commonly used Constructors of JColorChooser class:

- JColorChooser(): is used to create a color chooser pane with white color initially.
- JColorChooser(Color initialColor): is used to create a color chooser pane with the specified color initially.

Commonly used methods of JColorChooser class:

public static Color showDialog(Component c, String title, Color initialColor): is used to show the color-chooser dialog box.

---

15) JProgressBar class:

The JProgressBar class is used to display the progress of the task.

Commonly used Constructors of JProgressBar class:

- JProgressBar(): is used to create a horizontal progress bar but no string text.
- JProgressBar(int min, int max): is used to create a horizontal progress bar with the specified minimum and maximum value.
- JProgressBar(int orient): is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
- JProgressBar(int orient, int min, int max): is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used methods of JProgressBar class:

==========

1) public void setStringPainted(boolean b): is used to determine whether string should be displayed.
2) public void setString(String s): is used to set value to the progress string.
3) public void setOrientation(int orientation): is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants..
4) public void setValue(int value): is used to set the current value on the progress bar.

---

16) JSlider class:

The JSlider is used to create the slider. By using JSlider a user can select a value from a specific range.

Commonly used Constructors of JSlider class:

- JSlider(): creates a slider with the initial value of 50 and range of 0 to 100.
- JSlider(int orientation): creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
- JSlider(int min, int max): creates a horizontal slider using the given min and max.
- JSlider(int min, int max, int value): creates a horizontal slider using the given min, max and value.
- JSlider(int orientation, int min, int max, int value): creates a slider using the given orientation, min, max and value.

Commonly used Methods of JSlider class:

1) public void setMinorTickSpacing(int n): is used to set the minor tick spacing to the slider.
2) public void setMajorTickSpacing(int n): is used to set the major tick spacing to the slider.
3) public void setPaintTicks(boolean b): is used to determine whether tick marks are painted.
4) public void setPaintLabels(boolean b): is used to determine whether labels are painted.
5) public void setPaintTracks(boolean b): is used to determine whether track is painted.

Displaying graphics in swing:

java.awt.Graphics class provides many methods for graphics programming.
Commonly used methods of Graphics class:

public abstract void drawString(String str, int x, int y): is used to draw the specified string.
public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.

==========

public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.

public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.

public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.

public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.

public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.

public abstract void setColor(Color c): is used to set the graphics current color to the specified color.

public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

```
import java.awt.*;
import javax.swing.JFrame;

public class DisplayGraphics extends Canvas{

  public void paint(Graphics g) {
    g.drawString("Hello",40,40);
    setBackground(Color.WHITE);
    g.fillRect(130, 30,100, 80);
    g.drawOval(30,130,50, 60);
    setForeground(Color.RED);
    g.fillOval(130,130,50, 60);
    g.drawArc(30, 200, 40,50,90,60);
    g.fillArc(30, 130, 40,50,180,40);

  }
    public static void main(String[] args) {
    DisplayGraphics m=new DisplayGraphics();
    JFrame f=new JFrame();
    f.add(m);
```

```
================================================================================
        f.setSize(400,400);
        //f.setLayout(null);
        f.setVisible(true);
    }
  }
```

Displaying image in swing:

For displaying image, we can use the method drawImage() of Graphics class.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

```
  import java.awt.*;
  import javax.swing.JFrame;

  public class MyCanvas extends Canvas{

    public void paint(Graphics g) {

      Toolkit t=Toolkit.getDefaultToolkit();
      Image i=t.getImage("p3.gif");
      g.drawImage(i, 120,100,this);

    }
      public static void main(String[] args) {
      MyCanvas m=new MyCanvas();
      JFrame f=new JFrame();
      f.add(m);
      f.setSize(400,400);
      f.setVisible(true);
    }

  }
```

Example of creating Edit menu for Notepad:

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

  public class Notepad implements ActionListener{
  JFrame f;
```

==========

```
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;

Notepad(){
f=new JFrame();

cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");

cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);

mb=new JMenuBar();
mb.setBounds(5,5,400,40);

file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");

edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);

mb.add(file);mb.add(edit);mb.add(help);

ta=new JTextArea();
ta.setBounds(5,30,460,460);

f.add(mb);f.add(ta);

f.setLayout(null);
f.setSize(500,500);
f.setVisible(true);
}
```

```
==========================================================================================
   public void actionPerformed(ActionEvent e) {
   if(e.getSource()==cut)
   ta.cut();
   if(e.getSource()==paste)
   ta.paste();
   if(e.getSource()==copy)
   ta.copy();
   if(e.getSource()==selectAll)
   ta.selectAll();
   }

   public static void main(String[] args) {
      new Notepad();
   }
   }
```

---

17) BorderFactory Class

How to Use Borders

Every JComponent can have one or more borders. Borders are incredibly useful objects that, while not themselves components, know how to draw the edges of Swing components. Borders are useful not only for drawing lines and fancy edges, but also for providing titles and empty space around components.
Note:

Our examples set borders on JPanels, JLabels, and custom subclasses of JComponent. Although technically you can set the border on any object that inherits from JComponent, the look and feel implementation of many standard Swing components doesn't work well with user-set borders. In general, when you want to set a border on a standard Swing component other than JPanel or JLabel, we recommend that you put the component in a JPanel and set the border on the JPan

To put a border around a JComponent, you use its setBorder method. You can use the BorderFactory class to create most of the borders that Swing provides. If you need a reference to a border — say, because you want to use it in multiple components — you can save it in a variable of type Border. Here is an example of code that creates a bordered container:

==========

```
JPanel pane = new JPanel();
pane.setBorder(BorderFactory.createLineBorder(Color.black));
```

The Border API

The following tables list the commonly used border methods. The API for using borders falls into two categories:

- Creating a Border with BorderFactory
- Setting or Getting a Component's Border

Creating a Border with BorderFactory	
Method	**Purpose**
Border createLineBorder(Color) Border createLineBorder(Color, int)	Create a line border. The first argument is a java.awt.Color object that specifies the color of the line. The optional second argument specifies the width in pixels of the line.
Border createEtchedBorder() Border createEtchedBorder(Color, Color) Border createEtchedBorder(int) Border createEtchedBorder(int, Color, Color)	Create an etched border. The optional Color arguments specify the highlight and shadow colors to be used. The methods with int arguments allow the border methods to be specified as either EtchedBorder.RAISED or EtchedBorder.LOWERED. The methods without the int arguments create a lowered etched border.
Border createLoweredBevelBorder()	Create a border that gives the illusion of the component being lower than the surrounding area.
Border createRaisedBevelBorder()	Create a border that gives the illusion of the component being higher than the surrounding area.
Border createBevelBorder(int, Color, Color) Border createBevelBorder(int, Color, Color, Color, Color)	Create a raised or lowered beveled border, specifying the colors to use. The integer argument can be either BevelBorder.RAISED or BevelBorder.LOWERED. With the three-argument constructor, you specify the highlight and shadow colors. With the five-argument constructor, you specify the outer highlight, inner highlight, outer shadow, and inner shadow colors, in that order.
Border createEmptyBorder() Border createEmptyBorder(int, int, int, int)	Create an invisible border. If you specify no arguments, then the border takes no space, which is useful when creating a titled border with no visible boundary. The optional arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. This method is useful for putting empty space around your components.
MatteBorder createMatteBorder(int, int, int, int, Color) MatteBorder createMatteBorder(int, int, int, int, Icon)	Create a matte border. The integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. The color argument specifies the color which with

==========

	the border should fill its area. The icon argument specifies the icon which with the border should tile its area.
TitledBorder createTitledBorder(String) TitledBorder createTitledBorder(Border) TitledBorder createTitledBorder(Border, String) TitledBorder createTitledBorder(Border, String, int, int) TitledBorder createTitledBorder(Border, String, int, int, Font) TitledBorder createTitledBorder(Border, String, int, int, Font, Color)	Create a titled border. The string argument specifies the title to be displayed. The optional font and color arguments specify the font and color to be used for the title's text. The border argument specifies the border that should be displayed along with the title. If no border is specified, then a look-and-feel-specific default border is used. By default, the title straddles the top of its companion border and is left-justified. The optional integer arguments specify the title's position and justification, in that order. TitledBorder defines these possible positions: ABOVE_TOP, TOP (the default), BELOW_TOP, ABOVE_BOTTOM, BOTTOM, and BELOW_BOTTOM. You can specify the justification as LEADING (the default), CENTER, or TRAILING. In locales with Western alphabets LEADING is equivalent to LEFT and TRAILING is equivalent to RIGHT.
CompoundBorder createCompoundBorder(Border, Border)	Combine two borders into one. The first argument specifies the outer border; the second, the inner border.

Setting or Getting a Component's Border

Method	**Purpose**
void setBorder(Border) Border getBorder()	Set or get the border of the receiving JComponent.
void setBorderPainted(boolean) boolean isBorderPainted() (in AbstractButton, JMenuBar, JPopupMenu, JProgressBar, and JToolBar)	Set or get whether the border of the component should be displayed.

Examples that Use Borders

Many examples in this lesson use borders. The following table lists a few interesting cases.

Example	Where Described	Notes
BorderDemo	This section	Shows an example of each type of border that BorderFactory can create. Also uses an empty border to add breathing space between each pane and its contents.
BoxAlignmentDemo	How to Use BoxLayout	Uses titled borders.
BoxLayoutDemo	How to Use BoxLayout	Uses a red line to show where the edge of a container is, so that you can see how the extra space in a box

==========

		layout is distributed.
ComboBoxDemo2	How to Use Combo Boxes	Uses a compound border to combine a line border with an empty border. The empty border provides space between the line and the component's innards.

---

18) FileFilter Class

public abstract class FileFilter extends Object

FileFilter is an abstract class used by JFileChooser for filtering the set of files shown to the user.
A FileFilter can be set on a JFileChooser to keep unwanted files from appearing in the directory listing.

Constructor Detail

public FileFilter()

Method Detail

a) public abstract boolean accept(File f)
Whether the given file is accepted by this filter.

b) public abstract String getDescription()
The description of this filter. For example: "JPG and GIF Images"

Description

The java.io.File.list(FilenameFilter filter) returns the array of files and directories in the directory defined by this abstract path name that satisfy the given filter. All names are accepted if the given filter is null.

Declaration

Following is the declaration for java.io.File.list(FilenameFilter filter) method:
public String[] list(FilenameFilter filter)

Exception

SecurityException -- If a security manager exists and its SecurityManager.checkRead(java.lang.String) method denies read access to the file

==========

Program

```
package com;
import java.io.File;
import java.io.FilenameFilter;
public class FileDemo implements FilenameFilter {

  String str;
    // constructor takes string argument
  public FileDemo(String ext)
  {
    str="."+ext;
  }
  // main method
  public static void main(String[] args) {

    File f = null;
    String[] paths;

    try{
      // create new file
      f = new File("c:/test");

      // create new filter
      FilenameFilter filter = new FileDemo("txt");

      // array of files and directory
      paths = f.list(filter);

      // for each name in the path array
      for(String path:paths)
      {
        // prints filename and directory name
        System.out.println(path);
      }
    }catch(Exception e){
      // if any error occurs
      e.printStackTrace();
    }
  }
```

==========

```
  @Override
  public boolean accept(File dir, String name) {
    return name.endsWith(str);
  }
}
```

---

19) JFileChooser

The class JFileChooser is a component which provides a simple mechanism for the user to choose a file.

Class declaration

Following is the declaration for javax.swing.JFileChooser class:

public class JFileChooser extends JComponent implements Accessible

Class constructors

S.N.	Constructor & Description
1	JFileChooser() Constructs a JFileChooser pointing to the user's default directory.
2	JFileChooser(File currentDirectory) Constructs a JFileChooser using the given File as the path.
3	JFileChooser(File currentDirectory, FileSystemView fsv) Constructs a JFileChooser using the given current directory and FileSystemView.
4	JFileChooser(FileSystemView fsv) Constructs a JFileChooser using the given FileSystemView.
5	JFileChooser(String currentDirectoryPath) Constructs a JFileChooser using the given path.
6	JFileChooser(String currentDirectoryPath, FileSystemView fsv) Constructs a JFileChooser using the given current directory path and FileSystemView.

Class methods

S.N.	Method & Description
1	boolean accept(File f) Returns true if the file should be displayed.
2	void addActionListener(ActionListener l) Adds an ActionListener to the file chooser.
3	void addChoosableFileFilter(FileFilter filter) Adds a filter to the list of user choosable file filters.
4	void approveSelection() Called by the UI when the user hits the Approve button (labeled "Open" or

==========

	"Save", by default).
5	void cancelSelection() Called by the UI when the user chooses the Cancel button.
6	void changeToParentDirectory() Changes the directory to be set to the parent of the current directory.
7	protected JDialog createDialog(Component parent) Creates and returns a new JDialog wrapping this centered on the parent in the parent's frame.
8	void ensureFileIsVisible(File f) Makes sure that the specified file is viewable, and not hidden.
9	protected void fireActionPerformed(String command) Notifies all listeners that have registered interest for notification on this event type.
10	FileFilter getAcceptAllFileFilter() Returns the AcceptAll file filter.
11	AccessibleContext getAccessibleContext() Gets the AccessibleContext associated with this JFileChooser.
12	JComponent getAccessory() Returns the accessory component.
13	ActionListener[]getActionListeners() Returns an array of all the action listeners registered on this file chooser.
14	int getApproveButtonMnemonic() Returns the approve button's mnemonic.
15	String getApproveButtonText() Returns the text used in the ApproveButton in the FileChooserUI.
16	String getApproveButtonToolTipText() Returns the tooltip text used in the ApproveButton.
17	FileFilter[] getChoosableFileFilters() Gets the list of user choosable file filters.
18	boolean getControlButtonsAreShown() Returns the value of the controlButtonsAreShown property.
19	File getCurrentDirectory() Returns the current directory.
20	String getDescription(File f) Returns the file description.
21	String getDialogTitle() Gets the string that goes in the JFileChooser's titlebar.
22	int getDialogType() Returns the type of this dialog.
23	boolean getDragEnabled() Gets the value of the dragEnabled property.
24	FileFilter getFileFilter() Returns the currently selected file filter.
25	int getFileSelectionMode()

==========

	Returns the current file-selection mode.
26	FileSystemView getFileSystemView() Returns the file system view.
27	FileView getFileView() Returns the current file view.
28	Icon getIcon(File f) Returns the icon for this file or type of file, depending on the system.
29	String getName(File f) Returns the filename.
30	File getSelectedFile() Returns the selected file.
31	File[] getSelectedFiles() Returns a list of selected files if the file chooser is set to allow multiple selection.
32	String getTypeDescription(File f) Returns the file type.
33	FileChooserUI getUI() Gets the UI object which implements the L&F for this component.
34	String getUIClassID() Returns a string that specifies the name of the L&F class that renders this component.
35	boolean isAcceptAllFileFilterUsed() Returns whether the AcceptAll FileFilter is used.
36	boolean isDirectorySelectionEnabled() Convenience call that determines if directories are selectable based on the current file selection mode.
37	boolean isFileHidingEnabled() Returns true if hidden files are not shown in the file chooser; otherwise, returns false.
38	boolean isFileSelectionEnabled() Convenience call that determines if files are selectable based on the current file selection mode.
39	boolean isMultiSelectionEnabled() Returns true if multiple files can be selected.
40	boolean isTraversable(File f) Returns true if the file (directory) can be visited.
41	protected String paramString() Returns a string representation of this JFileChooser.
42	void removeActionListener(ActionListener l) Removes an ActionListener from the file chooser.
43	boolean removeChoosableFileFilter(FileFilter f) Removes a filter from the list of user choosable file filters.
44	void rescanCurrentDirectory() Tells the UI to rescan its files list from the current directory.

==========

45	void resetChoosableFileFilters() Resets the choosable file filter list to its starting state.
46	void setAcceptAllFileFilterUsed(boolean b) Determines whether the AcceptAll FileFilter is used as an available choice in the choosable filter list.
47	void setAccessory(JComponent newAccessory) Sets the accessory component.
48	void setApproveButtonMnemonic(char mnemonic) Sets the approve button's mnemonic using a character.
49	void setApproveButtonMnemonic(int mnemonic) Sets the approve button's mnemonic using a numeric keycode.
50	void setApproveButtonText(String approveButtonText) Sets the text used in the ApproveButton in the FileChooserUI.
51	void setApproveButtonToolTipText(String toolTipText) Sets the tooltip text used in the ApproveButton.
52	void setControlButtonsAreShown(boolean b) Sets the property that indicates whether the approve and cancel buttons are shown in the file chooser.
53	void setCurrentDirectory(File dir) Sets the current directory.
54	void setDialogTitle(String dialogTitle) Sets the string that goes in the JFileChooser window's title bar.
55	void setDialogType(int dialogType) Sets the type of this dialog.
56	void setDragEnabled(boolean b) Sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
57	void setFileFilter(FileFilter filter) Sets the current file filter.
58	void setFileHidingEnabled(boolean b) Sets file hiding on or off.
59	void setFileSelectionMode(int mode) Sets the JFileChooser to allow the user to just select files, just select directories, or select both files and directories.
60	void setFileSystemView(FileSystemView fsv) Sets the file system view that the JFileChooser uses for accessing and creating file system resources, such as finding the floppy drive and getting a list of root drives.
61	void setFileView(FileView fileView) Sets the file view to used to retrieve UI information, such as the icon that represents a file or the type description of a file.
62	void setMultiSelectionEnabled(boolean b) Sets the file chooser to allow multiple file selections.
63	void setSelectedFile(File file)

==========

	Sets the selected file.
64	void setSelectedFiles(File[] selectedFiles) Sets the list of selected files if the file chooser is set to allow multiple selection.
65	protected void setup(FileSystemView view) Performs common constructor initialization and setup.
66	int showDialog(Component parent, String approveButtonText) Pops a custom file chooser dialog with a custom approve button.
67	int showOpenDialog(Component parent) Pops up an "Open File" file chooser dialog.
68	int showSaveDialog(Component parent) Pops up a "Save File" file chooser dialog.
69	void updateUI() Resets the UI property to a value from the current look and feel.

---

20) JOptionPane

Introduction

The class **JOptionPane** is a component which provides standard methods to pop up a standard dialog box for a value or informs user of something.

Class declaration

Following is the declaration for **javax.swing.JOptionPane** class:

publicclassJOptionPane
extendsJComponent
implementsAccessible

Field

Following are the fields for javax.swing.JOptionPane class:

a) static int CANCEL_OPTION -- Return value from class method if CANCEL is chosen.
b) static int CLOSED_OPTION -- Return value from class method if user closes window without selecting anything, more than likely this should be treated as either a CANCEL_OPTION or NO_OPTION.
c) static int DEFAULT_OPTION -- Type meaning Look and Feel should not supply any options -- only use the options from the JOptionPane.
d) static int ERROR_MESSAGE -- Used for error messages.
e) protected Icon icon -- Icon used in pane.

==========

f) static string ICON_PROPERTY -- Bound property name for icon.
g) static int INFORMATION_MESSAGE -- Used for information messages.
h) static string INITIAL_SELECTION_VALUE_PROPERTY -- Bound property name for initialSelectionValue.
i) static string INITIAL_VALUE_PROPERTY -- Bound property name for initialValue.
j) protected Object initialSelectionValue -- Initial value to select in selectionValues.
k) protected Object initialValue -- Value that should be initially selected in options.
l) static string INPUT_VALUE_PROPERTY -- Bound property name for inputValue.
m) protected Object inputValue -- Value the user has input.
n) protected Object message -- Message to display.
o) static string MESSAGE_PROPERTY -- Bound property name for message.
p) static string MESSAGE_TYPE_PROPERTY -- Bound property name for type.
q) static int OK_CANCEL_OPTION -- Type used for showConfirmDialog.
r) protected int messageType -- Message type.
s) static int NO_OPTION -- Return value from class method if NO is chosen.
t) static int OK_OPTION -- Return value form class method if OK is chosen.
u) static string OPTION_TYPE_PROPERTY -- Bound property name for optionType.
v) protected Object[] options -- Options to display to the user.
w) static string OPTIONS_PROPERTY -- Bound property name for option.
x) protected int optionType -- Option type, one of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION or OK_CANCEL_OPTION.
y) static int PLAIN_MESSAGE -- No icon is used.
z) static int QUESTION_MESSAGE -- Used for questions.
aa) static string SELECTION_VALUES_PROPERTY -- Bound property name for selectionValues.
bb) protected Object[] selectionValues -- Array of values the user can choose from.
cc) static Object UNINITIALIZED_VALUE -- Indicates that the user has not yet selected a value.
dd) protected Object value -- Currently selected value, will be a valid option, or UNINITIALIZED_VALUE or null.
ee) static string VALUE_PROPERTY -- Bound property name for value.
ff) static string WANTS_INPUT_PROPERTY -- Bound property name for wantsInput.
gg) protected boolean wantsInput -- If true, a UI widget will be provided to the user to get input.
hh) static int WARNING_MESSAGE -- Used for warning messages.
ii) static int YES_NO_CANCEL_OPTION -- Type used for showConfirmDialog.
jj) static int YES_NO_OPTION -- Type used for showConfirmDialog.
kk) static int YES_OPTION -- Return value from class method if YES is chosen.

==========

S.N.	Constructor & Description
1	JOptionPane() Creates a JOptionPane with a test message.
2	JOptionPane(Object message) Creates a instance of JOptionPane to display a message using the plain-message message type and the default options delivered by the UI.
3	JOptionPane(Object message, int messageType) Creates an instance of JOptionPane to display a message with the specified message type and the default options
4	JOptionPane(Object message, int messageType, int optionType) Creates an instance of JOptionPane to display a message with the specified message type and options.
5	JOptionPane(Object message, int messageType, int optionType, Icon icon) Creates an instance of JOptionPane to display a message with the specified message type, options, and icon.
6	JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options) Creates an instance of JOptionPane to display a message with the specified message type, icon, and options.
7	JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue) Creates an instance of JOptionPane to display a message with the specified message type, icon, and options, with the initially-selected option specified.

S.N.	Method & Description
1	JDialog createDialog(Component parentComponent, String title) Creates and returns a new JDialog wrapping this centered on the parentComponent in the parentComponent's frame.
2	JDialog createDialog(String title) Creates and returns a new parentless JDialog with the specified title.
3	JInternalFrame createInternalFrame(Component parentComponent, String title) Creates and returns an instance of JInternalFrame.
4	AccessibleContext getAccessibleContext() Returns the AccessibleContext associated with this JOptionPane.
5	static JDesktopPane getDesktopPaneForComponent(Component parentComponent) Returns the specified component's desktop pane.
6	static Frame getFrameForComponent(Component parentComponent) Returns the specified component's Frame.
7	Icon getIcon() Returns the icon this pane displays.
8	Object getInitialSelectionValue() Returns the input value that is displayed as initially selected to the user.

==========

9	Object getInitialValue() Returns the initial value.
10	Object getInputValue() Returns the value the user has input, if wantsInput is true.
11	int getMaxCharactersPerLineCount() Returns the maximum number of characters to place on a line in a message.
12	Object getMessage() Returns the message-object this pane displays.
13	int getMessageType() Returns the message type.
14	Object[] getOptions() Returns the choices the user can make.
15	int getOptionType() Returns the type of options that are displayed.
16	static Frame getRootFrame() Returns the Frame to use for the class methods in which a frame is not provided.
17	Object[] getSelectionValues() Returns the input selection values.
18	OptionPaneUI getUI() Returns the UI object which implements the L&F for this component.
19	String getUIClassID() Returns the name of the UI class that implements the L&F for this component.
20	Object getValue() Returns the value the user has selected.
21	boolean getWantsInput() Returns the value of the wantsInput property.
22	protected String paramString() Returns a string representation of this JOptionPane.
23	void selectInitialValue() Requests that the initial value be selected, which will set focus to the initial value.
24	void setIcon(Icon newIcon) Sets the icon to display.
25	void setInitialSelectionValue(Object newValue) Sets the input value that is initially displayed as selected to the user.
26	void setInitialValue(Object newInitialValue) Sets the initial value that is to be enabled -- the Component that has the focus when the pane is initially displayed.
27	void setInputValue(Object newValue) Sets the input value that was selected or input by the user.
28	void setMessage(Object newMessage) Sets the option pane's message-object.
29	void setMessageType(int newType) Sets the option pane's message type.
30	void setOptions(Object[] newOptions)

==========

	Sets the options this pane displays.
31	void setOptionType(int newType) Sets the options to display.
32	static voidsetRootFrame(Frame newRootFrame) Sets the frame to use for class methods in which a frame is not provided.
33	void setSelectionValues(Object[] newValues) Sets the input selection values for a pane that provides the user with a list of items to choose from.
34	void setUI(OptionPaneUI ui) Sets the UI object which implements the L&F for this component.
35	void setValue(Object newValue) Sets the value the user has chosen.
36	void setWantsInput(boolean newValue) Sets the wantsInput property.
37	static int showConfirmDialog(Component parentComponent, Object message) Brings up a dialog with the options Yes, No and Cancel; with the title, Select an Option.
38	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType) Brings up a dialog where the number of choices is determined by the optionType parameter.
39	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) Brings up a dialog where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.
40	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) Brings up a dialog with a specified icon, where the number of choices is determined by the optionType parameter.
41	static String showInputDialog(Component parentComponent, Object message) Shows a question-message dialog requesting input from the user parented to parentComponent.
42	static String showInputDialog(Component parentComponent, Object message, Object initialSelectionValue) Shows a question-message dialog requesting input from the user and parented to parentComponent.
43	static String showInputDialog(Component parentComponent, Object message, String title, int essageType) Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.
44	static Object showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue) Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
45	static String showInputDialog(Object message)

==========

	Shows a question-message dialog requesting input from the user.
46	static String showInputDialog(Object message, Object initialSelectionValue) Shows a question-message dialog requesting input from the user, with the input value initialized to initialSelectionValue.
47	static int showInternalConfirmDialog(Component parentComponent, Object message) Brings up an internal dialog panel with the options Yes, No and Cancel; with the title, Select an Option.
48	static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType) Brings up a internal dialog panel where the number of choices is determined by the optionType parameter.
49	static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) Brings up an internal dialog panel where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.
50	static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) Brings up an internal dialog panel with a specified icon, where the number of choices is determined by the optionType parameter.
51	static String showInternalInputDialog(Component parentComponent, Object message) Shows an internal question-message dialog requesting input from the user parented to parentComponent.
52	static String showInternalInputDialog(Component parentComponent, Object message, String title, int messageType) Shows an internal dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.
53	static Object showInternalInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue) Prompts the user for input in a blocking internal dialog where the initial selection, possible selections, and all other options can be specified.
54	static voidshowInternalMessageDialog(Component parentComponent, Object message) Brings up an internal confirmation dialog panel.
55	static voidshowInternalMessageDialog(Component parentComponent, Object message, String title, int messageType) Brings up an internal dialog panel that displays a message using a default icon determined by the messageType parameter.
56	static voidshowInternalMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon) Brings up an internal dialog panel displaying a message, specifying all parameters.
57	static voidshowMessageDialog(Component parentComponent, Object message) Brings up an information-message dialog titled "Message".
58	static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType)

==========

	Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
59	static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon) Brings up a dialog displaying a message, specifying all parameters.
60	static int showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue) Brings up a dialog with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.
61	void updateUI() Notification from the UIManager that the L&F has changed.
62	static int showInternalOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue) Brings up an internal dialog panel with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.