Garbage collection: when no reference to an object exist, it is destroyed in Java. In c++ we need to use the delete operator to clear the memory.

Java code is converted into Bytecode. Java compiler converts Java code into bytecode which is then executed by Java Virtual Machine/Java Runt time system.

**Encapsulation**

Links data to the code that manipulates it. It provides another important feature called *access control*. We use *access modifiers.*

Main() is public because it is called by another function in the Java Run time system. When no access modifier is used, then by default the class is public within the package but not outside the package.

Access modifiers can be used for classes, fields and class functions.

Classes and packages are both means of encapsulation.

Java provides many layers of protection to allow fine grained control over the visibility of variables within classes, subclasses and packages.

Method Overloading:

Overloaded methods differ in the type of the parameters or the number of parameters. Changing return type alone is insufficient to distinguish two versions of a method.

PASSING ARGUMENTS TO FUNCTIONS

When we pass primitive data types, its call by value. When we pass variables that instantiate a class, it is call by reference.

**INHERITANCE**

A super class variable can reference a sub class object.

When a superclass variable refers a sub-class object, it can only access only those parts of the object defined by the superclass. Superclass has no knowledge of what a subclass adds to it.

Sub-class can refer to the immediate superclass using the keyword  *super*.

A subclass can call a superclass constructor by using the super keyword

super(int a,int b,int c); will call a superclass constructor. super() must always be the first statement in the subclass constructor.

class Box {

private int length, width, height;

Box(int l,int w,int h){length=l;width=w;height=h;}

Box(Box ob){width=ob.width;length=ob.length;height=ob.height;}

Box(){width=-1;length=-1;height=-1;}

}

class BoxWeight extends Box{

int weight;

BoxWeight(){

super();

weight=-1;

}

BoxWeight(int h,int l,int w,int weigh){

super(l,w,h);

weight=weigh;

}

BoxWeight(BoxWeight ob){

super(ob);

weight=ob.weight;

}

}

In the above code, BoxWeight cant access height, length, width because they are private fields of Box object. The only way to access or set them is to use the super keyword to access its constructor.

The ob variable in the Box(Box ob) constructor will refer to BoxWeight object. Since it doesnt know the definition of the sub-class, it can only access fields that are defined by it. So it will have access to width, length, height. The word super can also be used to refer to methods and fields in the superclass.

super.method\_name() will execute that method in the super class.

Method overriding happens when the name, return type and parameter, parameter type of both sub and super class matches. If they dont it is just method overloading.

The order of execution of constructors is superclass followed by subclass whether we use super or not. If we dont use super the default constructor of the superclass will get executed first.

Dynamic Method dispatch:

Call to an overridden function is resolved during run time than compile time. When a method is superclass is overridden by a method in subclass, it is not the type of the reference variable that determines which method to be executed- subclass or superclass? It is the type of object that we are referring to, that decides which method is executed. This is decided at run time not compile time.

Abstract modifier requires that certain methods be overridden by sublcass. These methods are referred to as subclasser responsibility. abstract keyword must then precede before the class keyword at the beginning of the class declaration. Abstract static methods and constructors cant be defined. Abstract classes cant be used to instantiate objects but can be used to refer objects

Sometimes we dont want a method to be overridden. Hence, we use the final keyword. Methods declared as final cant be overridden.

Sometimes you want to prevent a class from being inherited. To do this, precede the class declaration with final. Declaring a class final implicitly declares all its methods as final too. You cant declare a class both abstract and final.

Object is a class defined by java. All other classes are subclasses of Object. Object is a superclass of all other classes. Reference variable of type Object can refer to an object of any other class. Since arrays are implemented as clases, a variable of type object can also refer to any array.

Abstract classes dont support multiple inheritance. Interfaces however can help in multiple inheritance.

class A extends Aclass implements Iclass1, Iclass2 {

}

Abstract class can provide implementation of interface. An interface can extend another interface.

Interfaces can only have static and final variables. Abstract classes on other hands can have non-final and non-static variables too.

**Polymorphism**

Overridden methods allow java to support run-time polymorphism. One interface, multiple methods is how java implements run time polymorphism using method overriding.

**ABSTRACT CLASSES**

Abstract classes may or may not have abstract methods. But if there is one abstract method then the class must be defined abstract.

If a class is declared abstract it cant be instantiated.

**INTERFACE**

public interface name\_of\_interface{

}

Cant have constructors

All methods in interface are abstract

Fields can only be static and final.

Interfaces are implemented by classes.

An interface can extend multiple interfaces. A java class on other hand cant extend more than one parent class.

public interface hockey extends Events,Sports {

}

While overriding interfaces you cant change the method definition including the return type.

**PACKAGES**

Grouping of classes is called packages. Package is naming and visibility control mechanism. We can define classes in package which are not visible to code outside package.

package name\_of\_the\_package;

this should be the first statement in a java source file. All the classes declared within that file will belong to the specified package.

Package statement defines a name space within which a class is stored. Java uses filesystem directories to store packages.

.class files for any classes you declare to be a part of a package will be icnluded in that folder. Directory name must match package name exactly.

We can create a multileveled hierarchy in packages. To access them we simply write:

package pkg1[.pkg2[.pkg3]]];

for example

package java.awt.image;

the .class file will be stored in ./java/awt/image folder.

We cant rename a package without renaming the directory in which it is stored.

We can set a CLASSPATH environment variable which stores paths to different java packages.

Other ways to do it:

1) by default java will look for packages in the present working directory

2) we can specify the path to package using the -classpath option with java and javac

All the standard java classes are stored in a package called java. The basic language functions are stored in a package inside java package called java.lang. It is implicitly imported since other packages are useless without functionality provided by java.lang.

If two packages have classes with same name, then compiler will be silent as long as you dont use it. However, when you use it, you should specify to which package the class belongs.

**FINAL KEYWORD IN JAVA**

Final classes cant be inherited

Final methods cant be overridden

Final variables cant be changed or modified.

Constructors are not inherited . So marking them final is meaningless and not possible.

**STATIC KEYWORD IN JAVA**

Only

**NESTED CLASSES**

**ANONYMOUS CLASSES**

**FINALIZE**

Finalize is called prior to garbage collection.It is the counterpart of the constructor.

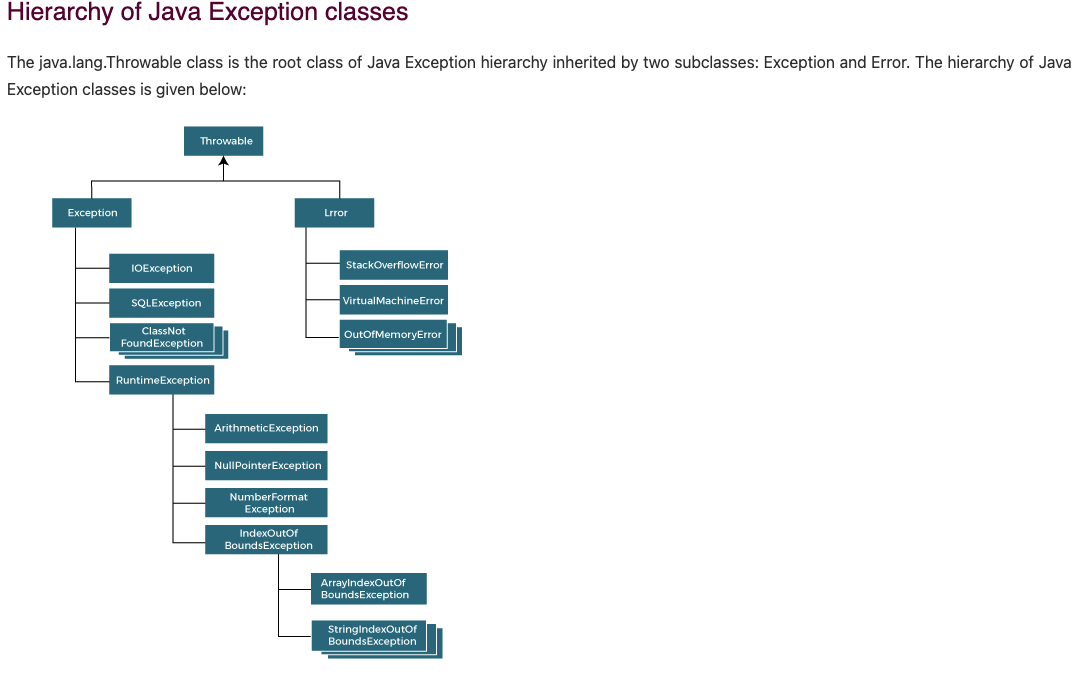
class A{

public void finalize(){

}

}

**EXCEPTION HANDLING**



Error are irrecoverable. Classes that inherit Runtime Exceptions are called unchecked exceptions.

Checked Exceptions are all the Exceptions except Runtime exceptions. They are called checked because the java compiler checks if the developer has handled such errors.

try{

}

catch{

}

finally{

//executed whether or not exception is handled or not

}

try can be without catch, but then it should be followed by finally. finally is executed regardless of whether exception is thrown or not. Finally block is used for cleanup functions like closing opened resources, releasing locks, freeing allocated memory.

The throw keyword is used for throwing custom Exceptions.

if(age<18)

throw new ArithmeticException(“person below 18 cant vote”);

class UserDefinedException extends Exception{

UserDefinedException(String str){

super(str);

}

}

class A{

public static void main(String[]args){

try{

throw new UserDefinedException(“this is the user defined exception”);

}

catch(UserDefinedException ude){

System.out.println(ude.getMessage());

}

}

}