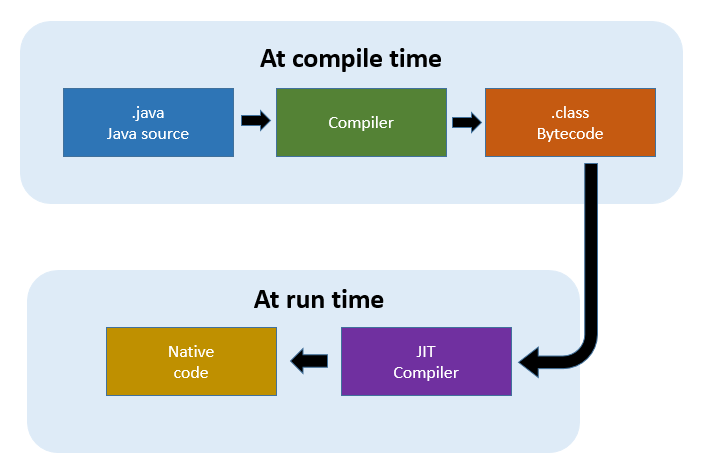
**JIT Compiler**

The JIT (just-in-time) compiler is a part of the JVM, it improves the performance of Java applications by compiling bytecodes to native machine code at run time

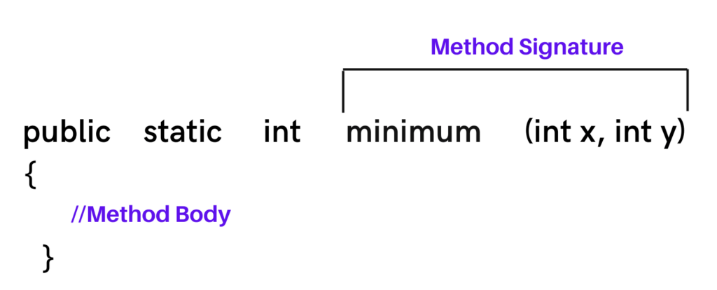


**Class Loader**

The Class loader is a part of the JRE that dynamically loads Java Classes into the Java Virtual Machine

**Method Signature**

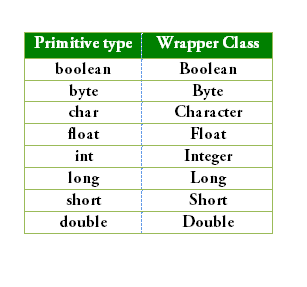
Method signature consists of method name and arguments



**if we don't declare the main method as static in Java,** the program gets compiled without compilation error, but at the time of execution JVM will not consider this method as the entry point of the program

**Wrapper Classes**

Wrapper Classes are used to convert primitive data types to objects



E.g., to convert primitive to wrapper

int x = 10;

Integer y = Integer.valueOf(x);

to convert the wrapper to a primitive

Integer x = Integer.valueOf(10);

int y = x.intValue();

**Default values of class variables**

Int = 0, Boolean = false, float = 0.0, String = null, char = \u0000

**Life Cycle of Object in Java**

Creation (using new keyword), initialization (using set method), use (using get method) and destruction (using finalize method)

**Local variables**

A local variable is a variable declared inside a method body; scope of the variable exists inside a method itself.

Local variables should be initialized to some value if not compiler will throw an error

static float *h*;

public static void main(String[] args) {

// **TODO** Auto-generated method stub

int a = 10;

System.***out***.println(*x*);

System.***out***.println(*str*);

System.***out***.println(*b*);

System.***out***.println(*ch*);

System.***out***.println(*h*);

System.***out***.println(a);

}

Here we assigned a value 10 for a local variable a.

**Instance Variable**

Instance variables are the one which is defined inside the class and outside the method and the scope of the variables exist throughout the class.

**Scanner**

The scanner is used to take input from a user.

Scanner in = new Scanner(System.in);

int x = in.nextint();

**String**

String is an object that represents the sequence of char values

String is immutable

**Why strings are immutable**

A String object is immutable, so you can't change its value after creation. This makes the String object thread-safe and so it can be safely used in a multi-threaded environment.

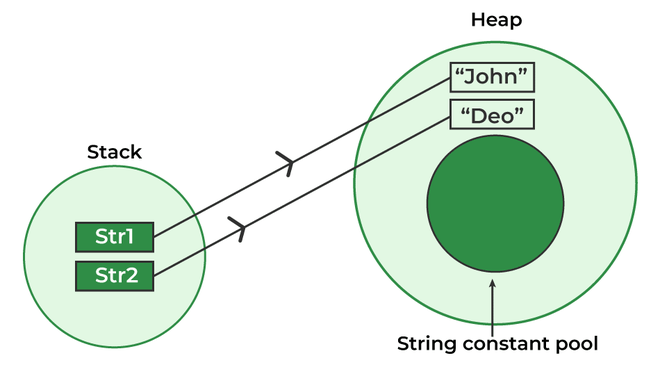
**Why Strings are immutable in Java**

A String in Java that is specified as immutable, as the content shared storage in a single pool to minimize creating a copy of the same value.

If we don't make the String immutable, it will pose a serious security threat to the application. For example, database usernames, passwords are passed as strings to receive database connections. The socket programming host and port descriptions are also passed as strings. The String is immutable, so its value cannot be changed. If the String doesn't remain immutable, any hacker can cause a security issue in the application by changing the reference value.

**String constant pool**

String pool is storage space in a Java heap memory where string literals are stored



**String Buffer**

String Buffer is like a string but can be modified (String buffer is mutable)

String Buffer is thread-safe and synchronized

**String Builder**

String Builder is mutable,

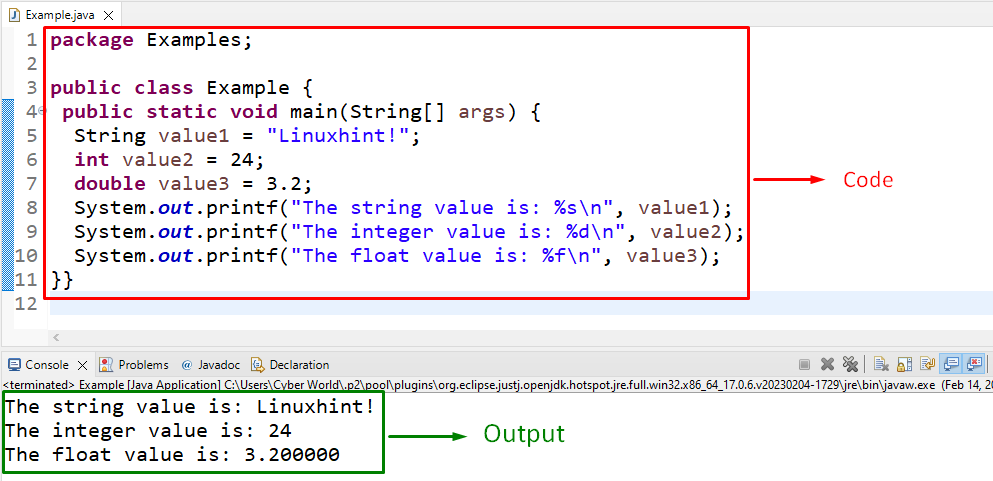
It is like String Buffer, but String builder is not thread-safe and not synchronized

Integer.parseint(number) -- to convert string to integer

Integer.toString(number) -- to convert integer to string

**Printf()**

The printf() method prints the output to the console with the use of various formatting commands



**Print()**

The print() prints the output to the console

**Println()**

The println() prints the output to the console and moves the cursor to a new line

**Covariant**

Covariant refers to the ability of a subclass method to override a superclass method

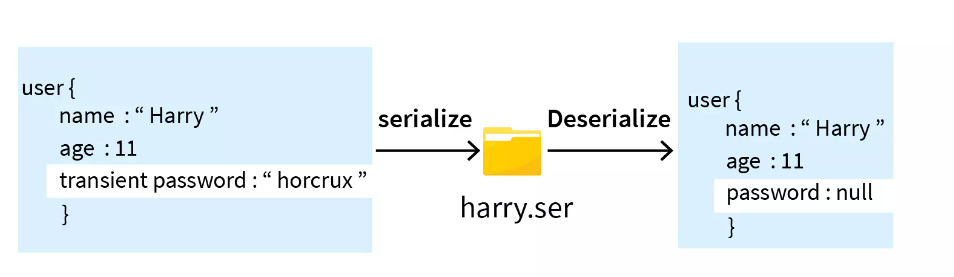
Covariant return type refers to the return type of an overriding method.

**Transient**

The transient keyword in Java is used to avoid serialization. At the time of serialization, if we don’t want to save the value of a particular variable in a file, then we use the transient keyword. When JVM comes across the transient keyword it ignores the original value of the variable and saves the default variable of that variable data type.

In order to exclude the instance variables from the serialization process, we use the transient keyword

static members cannot be serialized.



E.g., transient int x = 10;

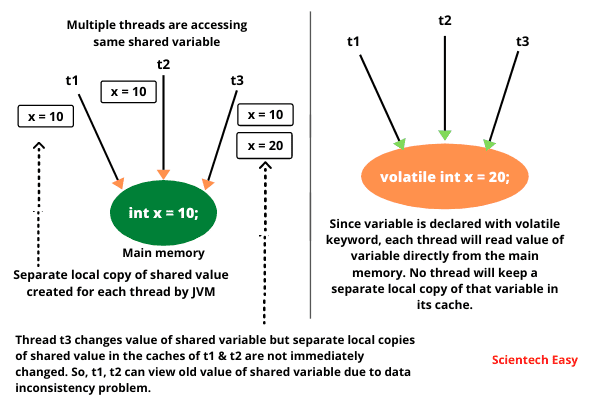
**Volatile**

Volatile keyword is used to modify the value of a variable by multiple threads.

A volatile keyword in Java tells the compiler the value of a variable may change at any time.

When we use the volatile keyword with a variable, all the threads always read the value of the variable directly from the main memory

Volatile keywords cannot be used with classes and methods



**Reference Variable**

A reference variable is a variable that holds the memory address of an object rather than the actual object itself.

A reference variable is a variable that points to an object of a given class, letting you access the value of an object.

E.g., {

A a1 = new A(); //here a1 is reference variable

}

**Interfaces**

Interface is a contract between client and service provider

Interfaces in Java is a blueprint of a class and contains static constants and abstract methods.

Variables are public static final and methods are public abstract

**Abstract method**

An abstract method is a method that doesn’t have a method body

**Concrete method**

A method with a body is a concrete method

**Abstract Class**

A class which is declared as abstract are known as abstract class. It can have both abstract methods and concretemethods. An abstract class is class which cannot be initiated by itself.

**Abstract**

Abstraction in Java is a process of hiding implementation details and showing only the functionality to the user.

**What is the difference between abstract classes and interfaces in Java**

Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods.

Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.

Abstract class can have final, non-final, static, and non-static variables. Interface has only static and final variables.

**super**

The super keyword in Java is a reference variable that is used to refer parent class objects.

**this**

this is a reference variable that is used to refer the current object through which the method is called.

**super()**

The super() in Java is a reference variable that is used to refer parent class constructors

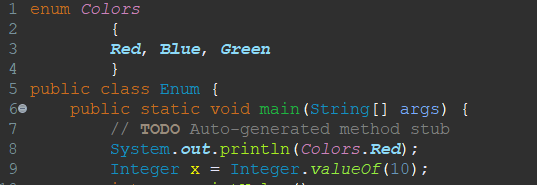
super can used to call parent class variables and methods, whereas super() is used to call parent class constructor only.

**this()**

this() is used to access one constructor from another where both constructors belong to the same class.

**Enum**

Enum in Java is a special data type that contains a fixed set of constants



**Default packages in Java**

java.lang.\* e.g., String, object

java.utli.\* e.g., collections

**Polymorphism**

Polymorphism is the ability to take more than one form

There are two main types of polymorphism, runtime polymorphism and compile-time polymorphism.

***Runtime polymorphism*** is achieved through method overriding

***Compile-time polymorphism*** is achieved through method overloading

***Method overriding*** is a type of dynamic polymorphism. Method overriding occurs when the method signature is the same in the superclass and the child class.

***Method overloading*** is a type of static polymorphism. In method overloading, we can define multiple methods with the same name but with different parameters



**Inheritance**

Inheritance means one class can extend to another class. So that the codes can reused from one class to another class. The existing class is known as the superclass/parent class and the derived class is known as subclass/child class

Inheritance in Java is a mechanism in which child object acquires all the properties and behaviours of a parent object.

***Advantage***: reusability of code.

Inheritance only applicable to public and protected members only.

Private members, static and final classes can’t be inherited

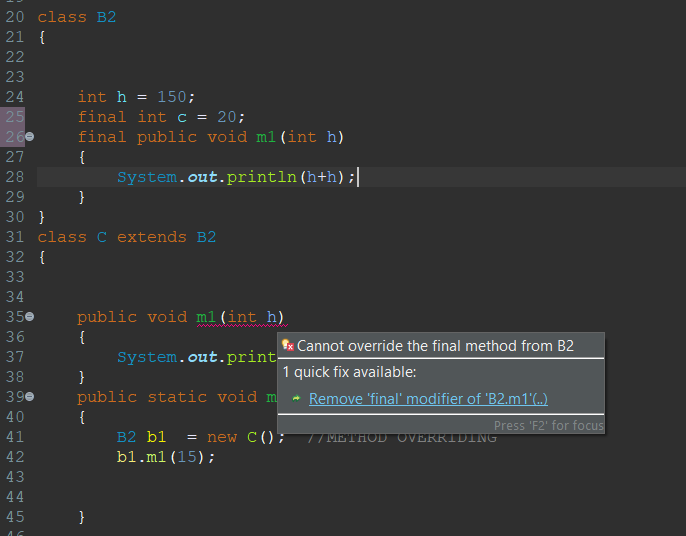
**Object class**

Object class in Java is the parent class of all the classes, it is present in java.lang package.

**Final**

In Java, the final keyword is used to indicate that a variable, method, or class cannot be modified/extended, and cannot be overridden by subclass.

A final method cannot be overridden by any subclasses. The final modifier prevents a method from being modified in a subclass.



**Encapsulation**

Encapsulation is a mechanism of wrapping code and data together into a single unit. It is to make sure that sensitive data is hidden from users.

Encapsulation is achieved by declaring the instance variables of a class as private.

<https://www.geeksforgeeks.org/encapsulation-in-java/>

**Garbage Collection**

Garbage collection in Java is an automated process of deleting code that’s no longer needed or used. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

**Exception**

An Exception is a problem that arises during the execution of a program.

***Exception handling***

Exception handling is a mechanism to handle runtime errors such as SQLException, ClassNotFoundException

Java provides 5 keywords that are used to handle the exceptions…try, catch, finally, throw, throws

***try***

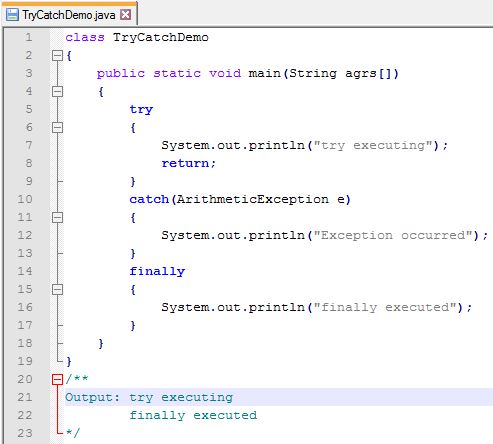
The try keyword is used to specify a block where we should place an exception code. We can’t use try block alone, it must be followed by either catch or final

***catch***

The catch block is used to handle the exception. It must be preceded by try block and it can be followed by finally block (We can’t use catch block alone)

***finally***

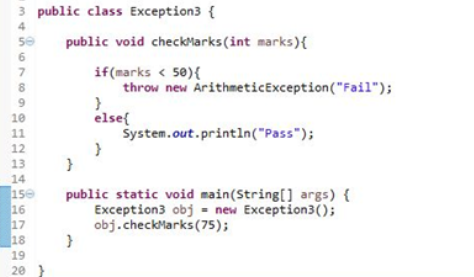
The finally block is used to execute the necessary code of the program. It is executed whether an exception is handled or not, (to free up resources)



***throw***

The throw keyword is used to create a custom error. It cannot throw multiple exceptions.

Throw is a keyword used inside a body of function. By using Throw, we can throw only one exception



***throws***

The throws keyword is used in a method signature and declares which exceptions can be thrown from a method.

The throws keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn’t throw an exception, it always used with method signature, we can declare multiple exceptions with throws keyword

Unchecked exception can be reproduced using throw but checked exception must use throws keyword followed by specific exception class name.

We should not declare throws and try catch block together.

Exception and error types are sub classes of class throwable

**What is the difference between checked and unchecked exceptions in Java**

checked exceptions are all compile time exceptions.

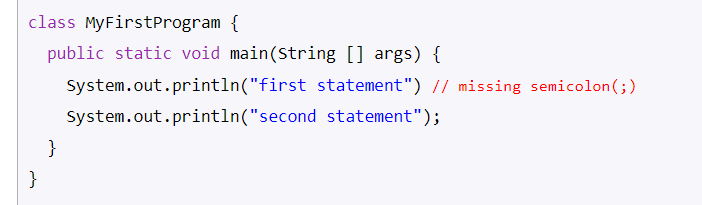
The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

unchecked exceptions are all runtime exceptions.

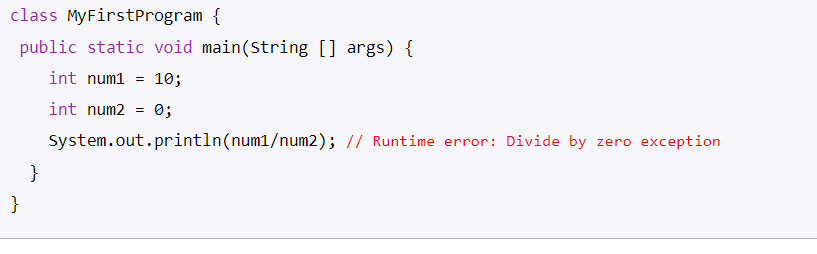
The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**Compile Time Error**

Eg: syntax errors



**Run Time Error**



**Different types of OOB exceptions in Java**

ArrayOutOfBoundException, StringIndexOutOfBoundsException

**Thread**

A flow of execution is called Thread. Every Java program has at least one thread called the main thread, the main thread is created by JVM

**MultiThreading**

Multithreading in Java is a process of executing two or more parts of the program simultaneously/running multiple threads simultaneously.

To create a thread Java provides a class Thread and an Interface Runnable both are located in java.lang package

***Thread Class***

The Thread class is a pre-defined class available in the java.lang package, which is used to create Threads, execute the threads, and manipulate the Threads.

***Runnable Interface***

The Runnable interface in Java is used to create threads. This interface contains a single method called run(). This method represents the code that will be going to execute whenever the thread starts.

public interface Runnable

{  
 public void run();  
}

The main difference between the Runnable interface and the Thread class in Java is that the Runnable interface is a functional interface, while the Thread class is a class. This means that the Runnable interface only has one abstract method, run(), while the Thread class has many methods, including start(), stop(), and yield().

We can create thread either by extending Thread class or implementing Runnable Interface

If we want to just create a thread it is recommended to use Runnable Interface, use Thread class for implementation of other thread functionalities as well.

**Collections**

Collections is utility class present in java.util.package. Collections provide a method for developers to perform certain basic operations on elements.

It contains only static methods

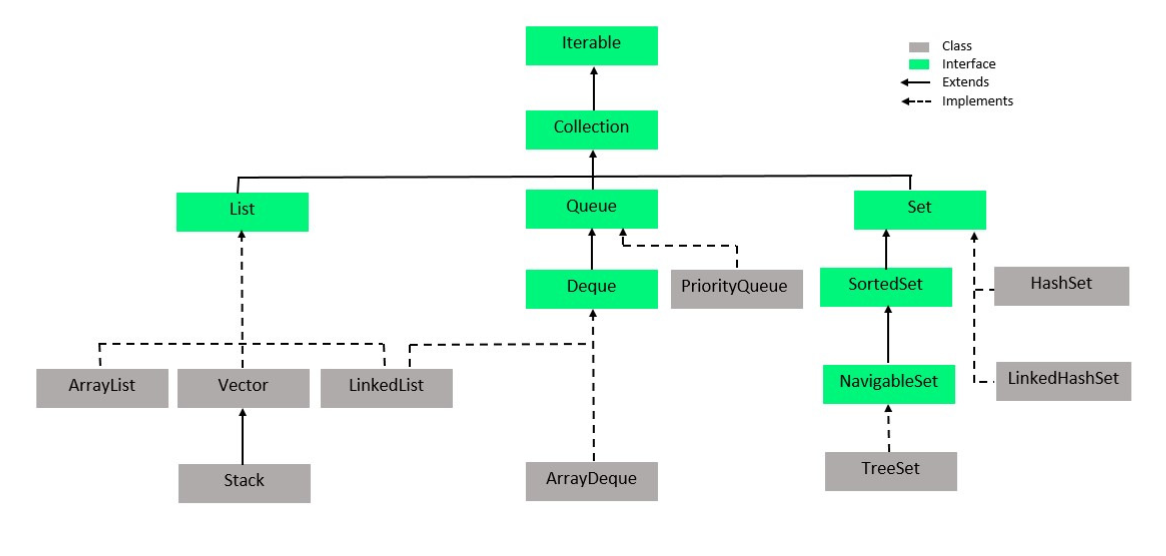
**Collection**

Collection is an interface present in java.util.package, where we group objects into a single unit.

The List, Set and Queue are the main sub interfaces of the collection interface. The map interface is also a part of the java collection framework, but it doesn’t inherit the collection of the interface.

It contains static method, also contains abstract and default methods

Collection is an interface whereas Collections is a utility class.



Collection is the interface where you group objects into a single unit. Collections is a utility class that has some set of operations you perform on Collection. The collection does not have all static methods in it, but Collections consist of methods that are all static.

**List**

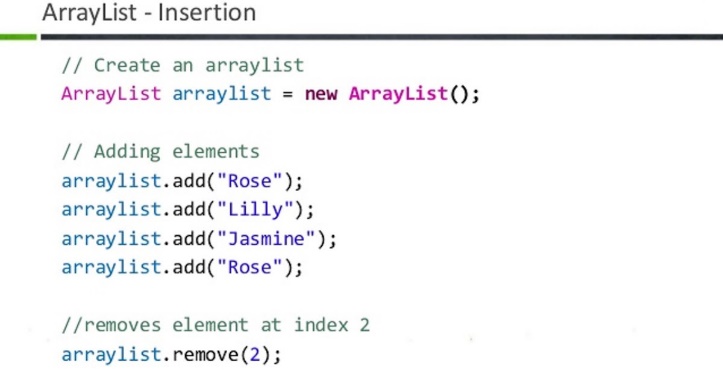
A List is an ordered Collection of elements which may contain duplicates.

It is an interface that extends the Collection interface. Lists are further classified into the following: ArrayList LinkedList Vectors

Vector is deprecated since Java 5.

**ArrayList**

ArrayList is the implementation of List Interface, it is similar to Array but there is no size limit in Arraylist you can add or remove the elements dynamically from the list.



***Disadvantage of ArrayList***

Never use Arraylist when you have to do lot of insertion and deletion as this will force arraylist to adjust the elements present in the list instead used LinkedList.

**How to convert Array to ArrayList and ArrayList to Array**

****

**LinkedList**

The Linked List in Java is a dynamic data structure whose size increases as you add the elements and decreases as you remove the elements from the list. LinkedList internally uses a doubly linked list to store the elements

Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory

Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory

List l = new LinkedList();

//List l = new ArrayList();

//l.add(a1);

l.add("John");

l.add(new ~~Integer~~(10)); l.add(10); l.add(a1.x);

l.add("tim");

l.add(2);

l.add(2.5);

l.add("l");

// for(int i = 0;i<l.size();i++)

// {System.out.println(l.get(i));}

// for(Object i:l)

// {System.out.println(i);}

ListIterator l1= l.listIterator();

while(l1.hasNext())

{System.***out***.println(l1.next());}

**Vectors**

Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.

Vector is synchronized and thread safe whereas ArrayList is not thread safe and not synchronized

 Vector v1 = **new** Vector();

        // Adding custom elements

        // using add() method

        v1.add(1);

        v1.add(2);

        v1.add("geeks");

        v1.add("forGeeks");

        v1.add(3);

**Queue**

Java Queue represents an ordered list of elements, Java Queue follows FIFO order to add or remove the elements

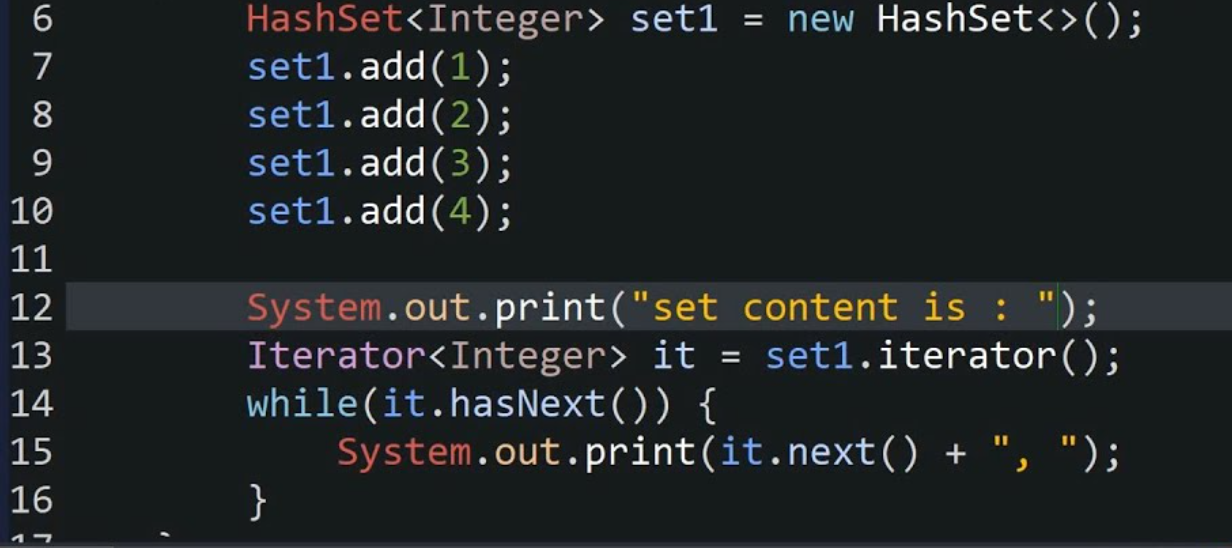
**Set**

It is an unordered collection of objects in which duplicate values cannot be stored.

**HashSet**

A HashSet is a collection of items where every item is unique

It cannot store duplicate values and HashSet doesn’t maintain any order of elements.



**LinkedHashSet**

Similar to the HashSet class, the LinkedHashSet class can also hold the unique elements only. But it maintains insertion order of elements.

LinkedHashSet<Integer> set = **new** LinkedHashSet<>();

        set.add(10);

        set.add(20);

        set.add(10);

        set.add(50);

        set.add(30);

        set.add(20);

        set.add(50);

**SortedSet**

The SortedSet class can also hold the unique elements only, and the elements are ordered by Ascending order, and it stores same data type only

SortedSet set = new TreeSet();

**TreeSet**

TreeSet is a child of SortedSet. SortedSet is an interface (it defines the functionality) and TreeSet is an implementation.

Set ts1 = new TreeSet();

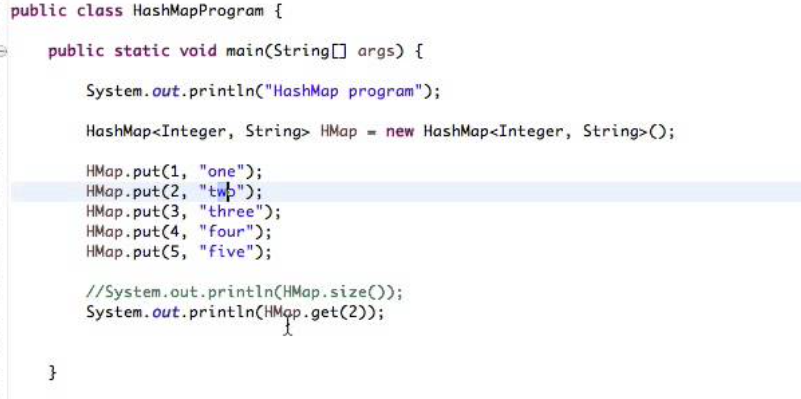
**Map**

Map in Java is an interface available in java.util package and it stores the data in key and value pairs. It does not allow duplicate keys.

**HashMap**

Java HashMap class implements the Map interface which allows us to store key and value pair

Java HashMap contains only unique keys, It may have one null key and multiple null values and it is not thread safe



**LinkedHashMap**

Java LinkedHashMap contains only unique keys, It may have one null key and multiple null values ,it is not thread safe and it maintains insertion order.

LinkedHashMap<Integer,String> hm=**new** LinkedHashMap<Integer,String>();

**SortedMap**

The SortedMap interface extends Map. It ensures that the entries are maintained in an ascending key order, no duplicates, and no insertion order

SortedMap<String, Double> map = new TreeMap<>();

**TreeMap**

Java TreeMap class is a red-black tree-based implementation. It provides an efficient means of storing key-value pairs in sorted order.

Java TreeMap cannot have a null key but can have multiple null values.

TreeMap<Integer,String> map=**new** TreeMap<Integer,String>();

**Hashtable**

Hashtable class implements a hashtable, which maps keys to values.

Java Hashtable class contains unique elements, It doesn't allow null key or value, It is synchronized/thread safe.

 Hashtable<Integer,String> hm=**new** Hashtable<Integer,String>();

**Comparable**

Java Comparable interface used to sort an array or list of objects by their natural order.

This interface is found in java.lang package and contains only one method named compareTo(Object).

**Comparator**

A comparator interface is used to order the objects of user-defined classes.

Comparator is present in java.util package.

Comparable in Java is an object to compare itself with another object, whereas comparator is an object for comparing different objects in different classes.

**toString()**

The toString() method converts an object into string and returns it.

**Finalize**

finalize() method in Java is a method of the Object class that is used to perform cleanup activity before destroying any object

**JDBC**

Java Database Connectivity (JDBC) is an application programming interface (API) for the Java programming language which defines how a client may access a database.

**Where to use Hashtable in real time**

Hashtable should be used in multithreading applications as it is thread safe and synchronized.

**Difference between == and .equals**

== is operator and .equals is method == is used for comparison for memory address. .equals is used for comparison of value

For example, if you have two String objects, "hello" and "hello", the equals() method will return true because the contents of the two objects are the same. However, the == operator will return false because the two objects are not the same object.

**intern()**

The intern() method is used to create an exact copy of a String object in the heap memory and stores it in the String constant pool.

Here is an example of how the code would work:

String s1 = "abc";

String s3 = "abc";

s3.intern();

System.out.println(s1 == s3);

O/P: true

**Does java support multiple inheritance? Give reasons?**

No, Java does not support multiple inheritance of classes. it can be unclear which method should be called and a compile-time error is generated.

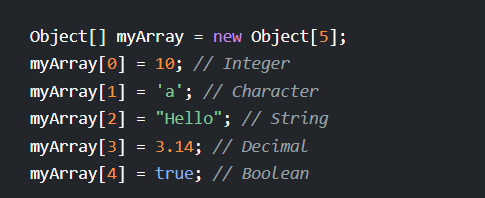
This is called Diamond problem

**Is Java a pure 100% Object Oriented Programming language?**

Java is not 100% object-oriented. This is because Java supports primitive data types, such as int, float, and char, which are not objects.

**How to assign different types of values as integer, character, string, decimal and boolean into the same array?**

In Java, you can assign different types of values to the same array by creating the array as an Object Class type. Here's an example:



**How to capture screenshot in Selenium WebDriver ?**

To capture screenshot in Selenium WebDriver use getScreenshotAs() from the TakesScreenshot interface

TakesScreenshot scrShot =((TakesScreenshot)webdriver);

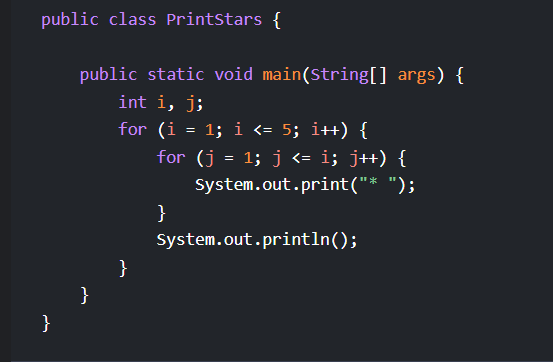
File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

File DestFile=new File(fileWithPath);

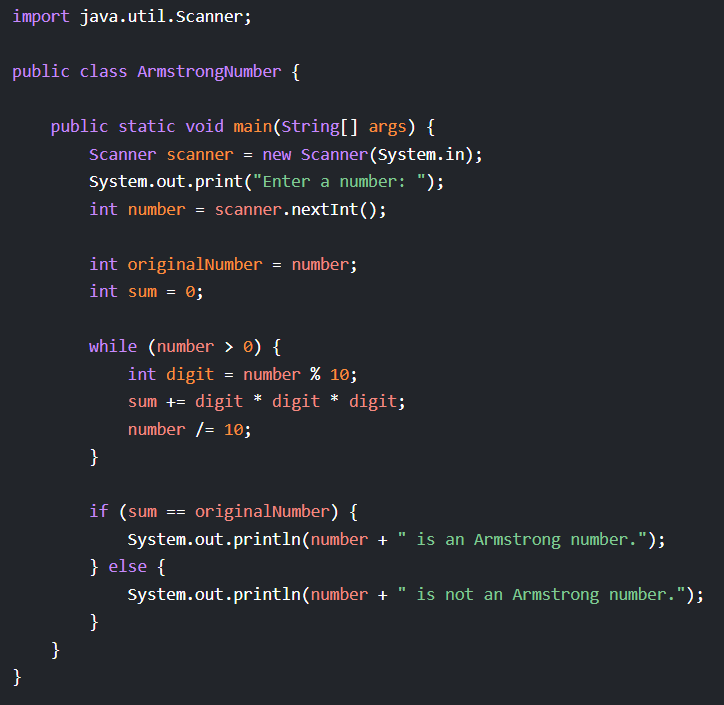
FileUtils.copyFile(SrcFile, DestFile);

Test cases to be automated Tests that need to be run with every build of the application (sanity check, regression) Tests that use multiple data values for the same actions (data driven tests) Complex and time-consuming tests

**Write a Java program to print stars using for loop, where the number of stars printed should be equal to the row number?**



**Write a Java program to check whether a given number is Armstrong ?**



**Synchronized block**

Synchronized block is used to prevent multiple threads from executing a portion of a code in a method at the same point in time.

**Explain SingletonDesign pattern in Java**

Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.

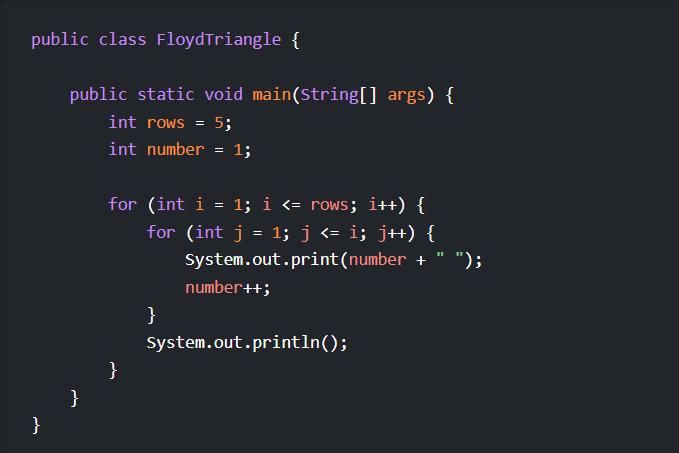
**What is the difference between collection and collections?**

The Collection is an interface whereas Collections is a class. The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.

**Is it mandatory to have the same name for Class name and Java file ?**

No, it is not mandatory to have the same name for Class name and Java file in Java. However, if you define a public class inside your java file, then the name of the file must be same as that of the public class.

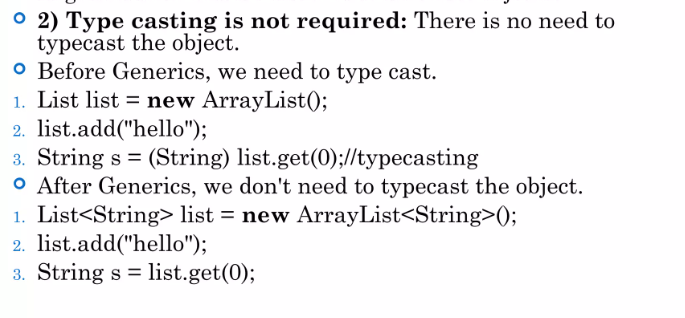
**Write a Java program to print Floyd’s triangle?**



# 

**What are generics in Java with example?**

Java Generics allows us to create a single class, interface, and method that can be used with different types of data (objects). This helps us to reuse our code. Note: Generics does not work with primitive types ( int , float , char , etc).



**Execute the java program without main method and only with static block**

It is possible to execute a Java program without the main method. However, after the static block executes, you will get an error saying no main method found.

**How to Convert Hashmap to Array**

To convert a HashMap to an ArrayList is to use the entrySet() method

