This notes will helps to know the java core in excellent but this is well to understand who are attended Jspiders training institute and source code is available who are attended the classes and this is a taste to join Jspider institute.

Java core

-Mr.Keshva EAR

"JAVA CORE"

Java programming involves 3 steps

- Coding
- Compilation
- Execution

1. Coding:

- Writing a java program using text editor or IDE (Eclipse, NetBeans) with **.java** extension.

2. Compilation

- Compilation is the process of translating a java file to class file.
- Compiler compiles java statements into byte codes.
- Bytes codes are saves with extension of .class file.
- .**class files are platform independent** file which runs on any platform.
- The .class file is created from the java source file.
- If any error occurs in the process of compilation is called **Compile time error.**

3. Execution

- Running the java class file is known as execution.
- The class file execution is done by **JVM (Java Virtual Machine).**
- The JVM utilities **JRE (JAVA RUNTIME ENVIRONMENT)** system libraries helps to run the byte codes.
- The .class file executed only when JRE is available.
- The java file is executed when file compiles successfully.
- If any error occurs at run time is called **Run time error.**

NOTE:

- ✓ To Compile java file by command line
  - **javac <filename>.java**
- ✓ to Run java file by command line
  - **java <classname>**

Variables:

Variables are used to store information, variable should follow 3 steps sequentially.

- Variable declaration
- Variable initialization
- Variable utilization

Variable Declaration

- Syntax: **type <variable name>**
- Type is data type to the type of information is stored.
- Variable name is any name starts with alphabet or underscore (_) not numeric part.
- Ex: int id;
  char grade;

Variable initialization

- Initialize a value or assign a value to the variable
- Ex : id=1234;
  grade='A';

Variable utilization

- Simply declaring and initializing is no use ,
- The utilization is need for every created variable
- Ex: System.out.println(id);
  System.out.println (grade);

Note:

- ✓ The java variables should be initialized prior to the utilization otherwise compilation gives compile time error.
- ✓ We can write variable declaration, initialization in a single line. And we can reassign variable multiple times.
- ✓ **Final** is a keyword in java which is to make constant variable. And it does not allow reassigning or reinitializing the variable. Ex : final float pi=3.142

Methods

Method has 2 types:

1. Static method
2. Non-static method (instance method)

- A method implements some particular task.
- Purpose of methods is reusability of code and reduces line of codes.
- It consists of opening and closing parenthesis in parenthesis it may have **arguments** or may not. Arguments are also called as parameters.
- Arguments is optional for methods, it may have multiple arguments.
- Methods will write once and can call multiple times. Methods should be defined when it is declared.
- Methods can return a value who calls it or to caller.

Control Statements

It has 2 types

1. Decision Control statements : 2 types
   a. IF
   b. SWITCH
2. Loop Control statements.
   a. FOR
   b. WHILE
   c. DO-WHILE

IF-ELSE

Syntax**: if** (condition)

{

<Do some business logic>

}

else

{

<Do some business logic>

}

NOTE:

- ✓ In this statements it may have only if statement, but **else** block should before of **if** block
- ✓ Based on the condition only one block will execute either if or else.

SWITCH

Syntax: **switch** (condition)

{

CASE 1:

<Some business logic>

Break;

CASE 2:

<Some business logic>

Break;

CASE N:

<Some business logic>

Break;

defaults:

<some business logic>

}

NOTE:

- ✓ Like multiples of IF-ELSE block but execute only one CASE among theses.
- ✓ In switch statement if there is no break statement it will execute all the case with default from which case its starts.

- ✓ If condition is not matching to any CASE statements it execute default statement.

FOR

Syntax: **for** (initialization; condition; increment/decrement)

```
{
    <some business logic>
}
```

WHILE

Syntax: **while** (condition)

```
{
    <Some business logic>
    Increment/decrement;
}
```

NOTE:

- ✓ Executes until condition becomes false.

DO-WHILE

Syntax: **do**

```
{
    <some business logic>
} while (condition);
```

NOTE:

- ✓ Executes until condition becomes true
- ✓ It executes minimum once time in its life time.

OOP'S CONCEPTS

Class:

- Class defines the state and behaviour of the objects.
- Class is a keyword, in java file can have multiple of java class file, while compiling compiler creates separate class file for each class body.
- JVM can starts execution from only one class at a time and the class must have main method and string array argument, otherwise JVM returns **Run time error**.
- Anything define within the body of class is known as members of a class.
- There are two types of members' 1.Static members and Non-static members.

Objects:

- Any entity which has state and behaviour is known as objects.
- The state defines the characteristics of object where as the behaviour defines the functionality of the object.
- The state defined by instance variable where as behaviour is defined by instance methods.

Variables:

Two types of variables

1. Primitive type variable.
2. Reference variable

Primitive type variable

- Used to store data only are declared using data type
- Ex: int i=10 //store only integer type data.
  Char grade='A' //store only single character type data.

Reference type variable

- Used to store objects.
- Class type variables are declared using class type/class name
- Syntax: class type
- Ex: class Apple //creating class by name Apple.
  Apple a1; //initialize a1 as type Apple class variable.

a1=new Apple (); //store only apple type object.

Static Members	**Non-Static Members**
Static members it may be methods or variables.	Non-Static members it may be methods or variables.
Static members have a keyword **static**	Except **static** keyword members is a non-static members or instance members.
Static members can access by class name. <Static members>.	Instance members can access by creating an object with reference variable.
Static members stored in static pool memory area.	Instance members stored in heap area.
JVM will loads first static members.	After static members Non-static members will load by JVM.
Static members can't be inherited.	Instance members can inherit, except private access members.
Static members can't be re-initialize.	Instance members can reinitialize.
Static members have only one copy of its in memory throughout life of execution.	Instance members can have multiple copies it's have to access by creating object.

Memory architecture:

To execute any program in computer 3 memory area come into picture as in the case of java programs execution it is also consist 3 part of memory area. Namely

1. Heap area:

- Used to store only objects, instance members with random allocation.

2. Stack area:

- Used to store local variables in this area.
- Used for execution of code in **L**ast **I**n **F**irst **O**ut manner.

3. Static pool:

- It is a part of heap, store only static members with random allocation.

Note: JVM loads first static members in static pool area.

Method:

- A method declared with pass by argument always accesses an object of the class type.
- When invoking such methods we must pass an object of the class either directly or passing the object reference.
- If a method declared with return type of class it should return object of the class.
- If a method didn't return any value it declared as **void** keyword
- A method can be defined in two fashion i.e. **Abstract method** and **Concrete method.**

Note:

- ✓ An object existing in the heap area without any reference variable is known as de-reference object.
- ✓ De-reference object consumes heap memory but can't access its members since it doesn't have reference. To make de-reference object use **null** to the object reference for removing all the object reference.
- ✓ Whenever the garbage collector is called on priority it removes de-reference object from heap.
- ✓ Making space for other object creation we can call garbage collector anywhere in the program execution using **System.GC** statement.

Constructor:

- Constructors are special members of java class used for object creation and instance variable initialization.
- Each and every java class must define a constructor; create an object of the class.
- If a class is defined without constructor compiler finds a constructor it is known as **default constructor**.
- A default constructor should not have any arguments.

Constructor is similar to methods but its have certain special features.

- The constructor name is **same** as class name.

- The constructor should not have any **return type**
- Defining a constructor with arguments is known as **parameterized constructor.**
- A programmer can define a constructor by following the syntax:

```
Construcorname (args)
{
    ----------------
    ----------------

 }
```

Note:

- ✓ Defining a multiple constructor with different arguments is known as **constructor overloading**.
- ✓ The arguments must differ either in terms of argument type or argument length.
- ✓ Overloaded constructors are invoked based on the arguments, when we create an object with the different initialization then we will go for overloaded constructor.

this :

- Java provides a special keyword by name with "this" it is used to **refer the instance members of the current class.**
- This keyword should be used only in constructor body or instance method body.
- It also used in differentiating the instance variable from local variable or arguments.

this ():

- Make constructor call.
- Call arguments constructor.
- Used in constructor body.
- Must be first statement in call constructor.
- Recursive constructor call is not allowed.

Note:

- ✓ In a static context static members of a current class can be referred directly where as the non-static members can't be referred in the static context.
- ✓ In a non-static context both static and instance members of current class can be referred directly, whenever instance members are referred compilers will write "this" keyword implicitly.

Composition:

- An object can have reference to another object this is known as **composition**. Where an object is composed of several object.
- The composition is also known as **"has-a"** relationship.
- A **"has-a"** relationship is achieved by defining reference variable in a class body. Reference variable of an object can be instance/static variable reference.
- The instance variables are referred by the object of class and static variable is referred by the name of the class.
- **System.out.println ()** is best example for composition how
  System: It is a built in library class.
  out : Static reference of output stream type of ,member of system class.
  println():It is overloaded instance method of OutputStream type.

Note:

- ✓ In the above statement said composition how by naming convention.
- ✓ Using naming convention we can decide the statement status.

Inheritance:

- Inheritance is OOP'S principle where one class acquire the properties another class. In java class can inherit the properties of another class by using **extends** keyword.
- The class from where members are inherited is known as **superclass** (base class or parent class).
- The class to which members are inherited is known as **subclass** (derived class or child class).

- Sub class can inherit only instance members of super class.
- Inheritance is used for code **reusability** and **extension ability**.
- There are four types of inheritance
    1. Single inheritance.
    2. Multilevel inheritance.
    3. Multiple inheritance.
    4. Hierarchical inheritance.

Note:

- ✓ Static members can't be inherit from super class.
- ✓ Private members can't be inherit.
- ✓ Final class can inherit but do not change the members' implementation in subclass.

Super:

- Using **super** keyword to call immediate super class constructor
- Implicit for default constructor.
- Super keyword support for inheritance but this keyword is within class body only.
- It's certain rules are similar to this keyword.

super ():

- A Sub class constructor can make a call to its super class constructor by using super () statement.
- The super statement should be the first statement in the body of sub class constructor.
- Using super statement sub class can call either default or parameterized constructor of super class constructor.
- Using super statement implement **constructor chaining;** means subclass constructor calling to its super class constructor, that super class constructor calling to its super class constructor is known as constructor chaining. Constructor chaining must happen during inheritance.
- The subclass constructor can make call to super class constructor either implicitly or explicitly.
- Implicit call done by compiler only for default constructor, when as explicit call can we made any constructor of super class.

Note:

- In a constructor body multiple super () statement or multiple this() statement or super() and this() statement is not allowed. Only one either this() or super()

Diamond problem:

Each and every java class must have super class. If we don't define super class, the compiler defines a super class by name **Object.**

Object is a super most class in java class hierarchy. Each and every java class must inherit the members from Object class either implicitly or explicitly. Java doesn't support multiple inheritance because the subclass constructor can't make call to the more than one super class constructor and also leads an ambiguity is known as diamond problem.

Method overloading:

- Defining multiple methods with same method name and different arguments is known as method overloading. The arguments must differ either in argument type or in an argument length.
- We can overload both static and non-static method in a class. Subclass can overload the inherited non-static methods only not static methods.
- The overloaded methods are invoked based on the arguments type through method overloading in this we can achieve **compile time polymorphism**.
- When developing an application if we go for method overloading.
- E.g. : A flight reservation application the book flight information can be viewed either by book id or flight number or date of travel or source and destination.

Method overriding:

- Inheriting methods from super class and changing its implementation in the subclass according to the specification of subclass is known as method overriding.
- To override the methods inheritance "**is a"** relationship is required. When a subclass overrides super class method then the subclass shouldn't change the signature of the inherited method
- After overriding method in the subclass overloading implementation will be available. The subclass can override as well as overload inherited methods.
- Overriding methods will achieve **run time polymorphism.**
- The subclass can't be override when

1. Static methods because static methods will not be inherited to subclass.
2. Final instance methods because the final keyword indicates that the definition can't be changed further.
3. Private instance methods because private are no accessible in subclass.

Static hiding: Static hiding is nothing but a subclass have static method which is same as in superclass.

Casting:

- Casting is converting one type date to another type is known as type casting. That type casting can be classifieds into 2 types those are
    1. Primitive typecasting
    2. Class casting.

1. Primitive type casting:

- Type matching statements

```
int i=2;
double d=5.3;
```

- Type mismatching statements

```
int i=2.3;
double d=5;
```

- Type casting

```
int k= (int) 59.99;
system.out.println (k)        //59
double d= (double) 34;
system.out.println (d);       //34.00
```

- Casting one data type to another data type is known as primitive type casting.
- Primitive type casting can be done in 2 ways.
    1. Widening: Casting lower data type to any higher data type is known as widening, widening can be done by implicitly and explicitly.
    2. Narrowing: Casting higher data type to any of lower data type is known as narrowing. Narrowing should be done explicitly in the program, when narrowing happen precision will be loss.

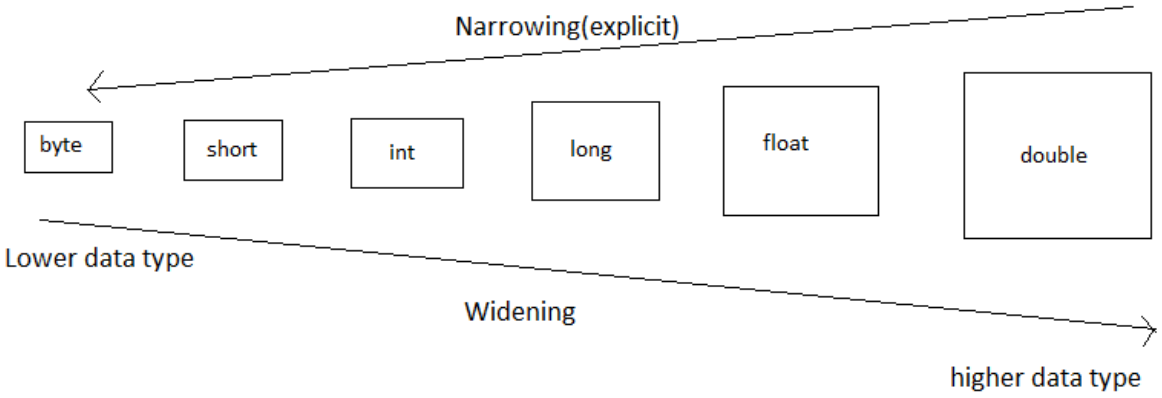


Fig. Primitive type casting.

2. Class casting:

- Type matching:
  A a1= new A ();
  B b1= new B ();
- Type mismatching :
  A a2=new B ();
  B b2= new A ();
- Class casting:
  A a2= (A) new B ();
  B b2= (B) new A ();
- Casting one class type to another class type is known as class casting.
- Class casting is possible only if classes having **"is a** "relationship (inheritance).
- Class casting can be done in 2 ways
  1. Up casting: Casting sub class type to super class type is known as up casting. Whenever the up casting is done the subclass object should show only the behaviour of super class or to the class which is casted.
     Up casting can be done either implicitly or explicitly.
     Sub class object can be casted to any level of superclass.
  2. Down casting: The down casting is possible only when objects are already up casted otherwise throws an exception.
     Down casting should be done explicitly in the code.

Polymorphism:

- Polymorphism is an object showing different behaviour at different stages of life cycle is known as **polymorphism.**

- There are two types of polymorphism:
    1. Compile time polymorphism.
    2. Run time polymorphism.

1. Compile time polymorphism:
   In this a method declaration is binded to the method definition by compile time by compiler during compilation this type of binding is known as **static binding** or **early binding** or **compile time binding**. Ex: overloading static, final, and private methods.
   Overloaded method is a compile time polymorphism.
2. Run time Polymorphism:
   In this a method declaration is binded to the method definition by the JVM based on the object created, this type of binding is known as **dynamic binding** or **late binding** or **run time binding.** Ex: method overriding is example for Run time polymorphism.
   To achieve run time polymorphism we have to fulfil the following:
    - Inheritance.
    - Overriding method.
    - Up casting.

Abstraction:

A method is designed in two manners either Abstract or Concrete.

- A method defined with body is known as **concrete method**, the concrete method can be called for execution and whenever it is called it executes the statement defined in the body of the method. While creating an object if RHS=LHS then it should be a concrete class.
- Any method defined without any body is known as **abstract method.** An abstract method should be declared **"abstract"** keyword.
- If any one of method is abstract in a class that class should be declared as abstract keyword is known as **abstract class**.
- In abstract class there is **0 or 1 or more abstract** methods.
- We can't create an object of abstract class and hence it is not possible to refer the instance members of abstract class. In an abstract class we can define:
    1. Only concrete method.
    2. Only abstract method.
    3. Both concrete and abstract method.
- An abstract class need not to have abstract method where as any class having abstract method should be declared as abstract.

Note:

- ✓ A class can inherit the abstract method from an abstract class, in such case the sub-class should be declared either abstract or sub-class should made concrete by defining all inherited abstract methods.
- ✓ Abstract methods of an abstract class can be define in any level of abstract class. Only those sub class object can be created which define in all the inherited abstract class.
- ✓ The abstract keyword can't be combine with following combination.
  1. Abstract static void test();
  2. Abstract final void test();
  3. Abstract private void test();

Java interface:

- Since abstract class allows us to define the class body with only concrete methods, it is known as not yet pure abstract body because a sub-class inherits the abstract class it becomes concrete without overriding the methods to achieve pure abstract body we go for **"java interface".**

Java class/class type	**Java interface/interface type**	Java enumeration/enum type

Class class_name
{
➢ Variables
• Static
• Non-static
➢ Constructors
➢ Methods(abstract/concrete)
• Static
• Non-static
}
	Interface interface_name
	{
	➢ Only static final variable.
	➢ No constructors.
	➢ Only instance abstract method.
	➢ Members of interface have access public.
	}
		Enum enum_name
		{
		➢ Only constant variables
		• Final
		• Static
		}

- The java interface has the above properties.

Note:

- ✓ Abstraction is one of the OOP'S principle which is used to **hide the class implementation.**
- ✓ Hiding the implementation of an object from its usage is known as abstraction. The object implement is access through the interface type reference.

- ✓ Abstraction can be achieved by following :
  1. **Generalize** the object behaviour in an interface.
  2. Provide specific implementation in a sub-classes.
  3. Refer the sub-class implementation through interface reference.
  4. The abstraction helps to develop maintainable code and easy to enhance.
  5. Using abstraction we can achieve **loose coupling**.

Loos coupling: Development of an application having a separate part of object creation and initialization, object design and implements and object usage. Which is end-user can't change or not able to modify the code. If developer has to change the code not impact on the end-user application.

Java API'S: An API is java library which is collection of packages which provides specific functionality. API's are developed to integrate any two software systems. A one software system can use the features of another software system by using API'S. The API'S are developed in such a manner any changes in the system will not have much impact in another system.
Example:

If a java application has to interact with database then it has to use JDBC API. So that, if the database is changed no need to change the application.
If a java application has to interact with MS-office then we have to use POI API.

Access specifiers:

- Private
- Package level
- Protected
- Public

Encapsulation:

- Encapsulation is a process of protecting the members of class body. The access to the class members can be achieved by using **access specifiers.**
- **The private** members has visibility up to the class level, it can be access only within the class body where it is declared. The private members provides very good restriction.
- **The Package** level access members has a visibility up to package level, it can be access within the same class body as well as from the other class body which belongs to same package. The package level members can't be access from outside the package.

- **The Protected** members has a visibility up to package level but it can referred from outside the package only though inheritance ("is a" relationship).
- **The Public** members has visibility outside the package and it can be access from any class body.

Rules access specifiers:

Rule 1:

- Defining class in a class body is known as inner class. An inner class can have any of the 4 access specifiers.
- The outer class can have public either or default. It can't have private as well as protected.

Rule 2:

- A java file can have any number of class body or interface body, only one of the body can have public access specifier, in such case the file name should be same as public class name or interface name.
- If all the class body or interface body have default access then the file name can be any name.
- If we have to have all the class public then we should define in an individual java file.

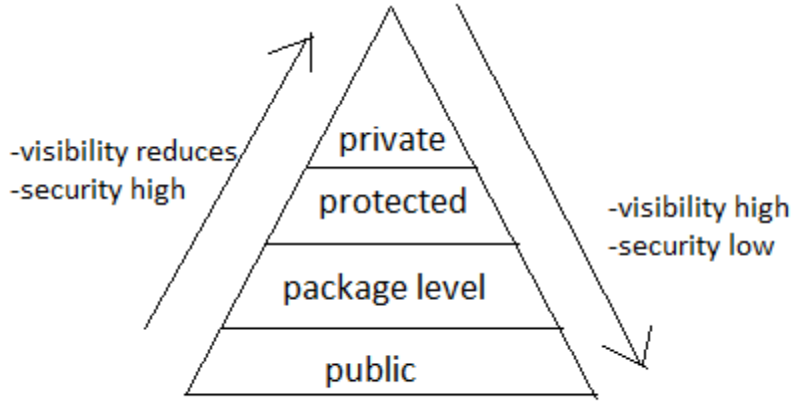


Fig.Access specifiers

Rule 3:

- Constructor defined by compiler is known as default constructor, the access of default constructor will be same as the class body meaning if the class is public the default constructor access is also public.

- If the programmer defines a constructor then programmer can set any of the 4 access specifiers.

Rule 4:

- When sub class overrides inherit method of super class. The subclass has permission to increase the visibility of inherited method but doesn't have the permission to reduce visibility of inherited method.

Rule 5:

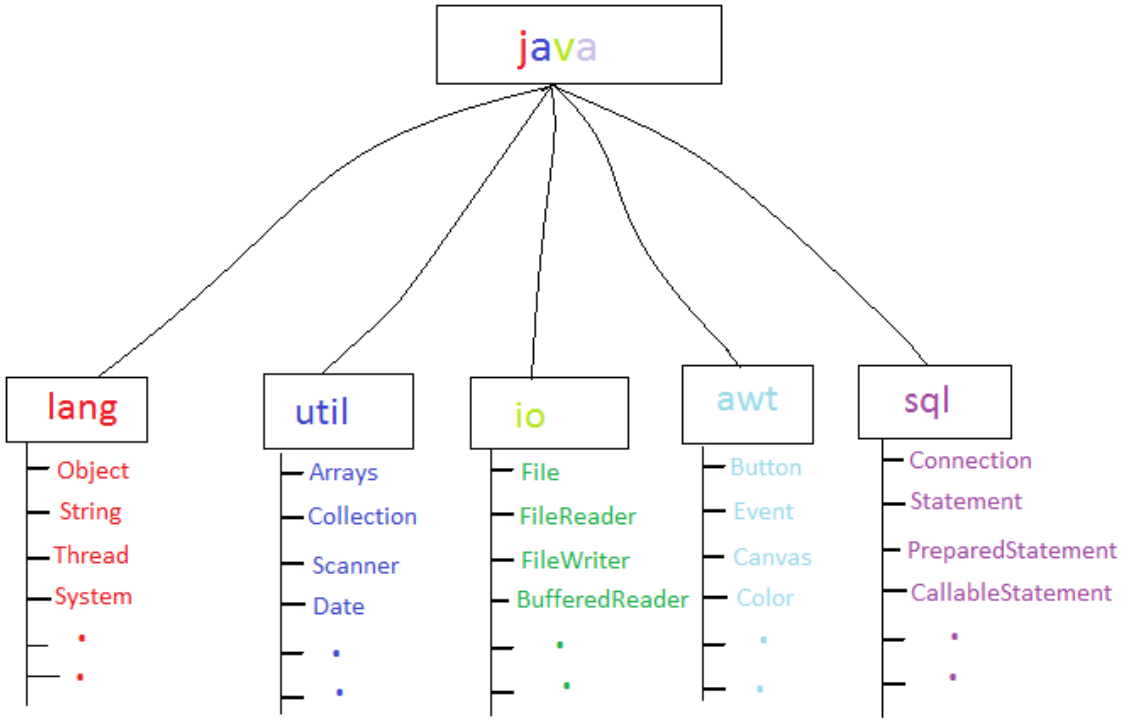
- The default access in the class body is package level where as the default access in the interface is public. Whenever the class implements an interface method the class should explicitly keep the access as public. Because if class doesn't provide access then it will be considered as default which is package level, and it has low visibility compared with inherited method which is public.

Getters-Setters:

- Defining a java class with private data members, public constructor, getters and setters method is known as **java bean class.**
- Java bean class are used for accessing information from database.
- The getters methods are developed to provide read access to the private data members, it is also known as accesser.
- The setter methods are developed to provide wright access to the private data members, it is also known as mutater.
- Both getter and setter methods should be public access specifier.

Packages:

Packages is a collection of java files which performs specific functionality. package is like a folder which consist of classes and interface in a systematic manner this is for good practice for professionals. In real world application are developed in this manner to easy maintenance of the each module. To create package is simply use package keyword with name of the package at first statement in the program. There is a standard industry rules to make package name like these: **com.organization_name.application_name.sub-application_name.sub-module.filename.**



Methods of Object package:

1. toString():-
   - to return string representation of the object ,return fully qualified class name .Ex: *java.lang.Object*
   - Return type is string, access specifier is public.
   - This method can be override in sub-class.
2. hashCode():-
   - To return hash code value of the object.
   - Return type is int, access specifier is public.
   - This method can be override in sub-class.
3. equals():-
   - Used to compare an object itself with another object.
   - Return type is Boolean, access specifier is public.
   - This method can be override in sub-class.
   - Argument of object is class type.
4. finalize():-
   - This is called by garbage collector to check whether the object is de-referenced or not.
5. Clone():-
   - Is used to clone the object means exact replica of the object.

Note:

- ✓ There are number of packages, above shown are popular and used frequently.
- ✓ If toString () is overridden, it is automatically returns the output by creating object.
- ✓ Hashcode will create unique id for every created object, reference type variable will not create unique id.

String class:

String is a set of character or array of character, string literals are type of object. String is not a data type it is class object type.

String class features

- Used to store string literals.
- Available in *java.lang package.*
- It is a final class.
- It is immutable class.
- toString () override to return string literal.
- hashCode () override to return hashcode value based on string literal.
- equals () override to compare strings based on string literal.
- String class constructor is overloaded.
- String class contain useful methods like strcmp, strlen, strconcat, etc.
- String type object can be created in two ways
  1. By using **"new"** keyword.
  2. By using **double quoted** string literals directly.
     When using double quoted string object creation string will not create an object of new, it's only pointing to the existing object.

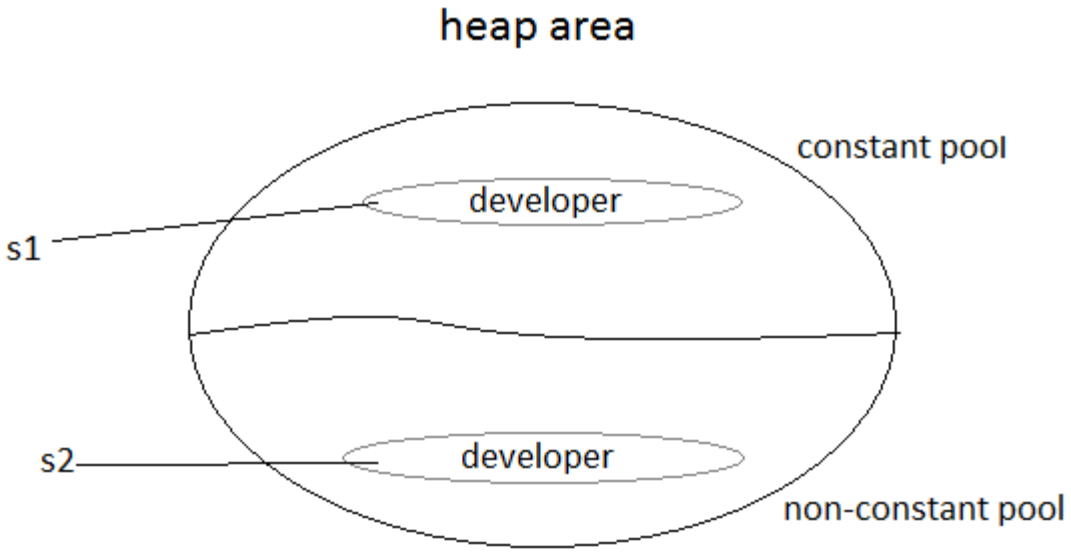
In string object creation object stored in memory based on the type of the object creation. Heap area consist of 2 areas for storing string literals.

1. Constant pool:-
   - Which is created an object of the string by using double quoted directly are stored in the constant pool memory area
   - It doesn't allow to duplicate the object.
   - Hashcode () and equals () are return based on string literals. If string literals are same, hashcode () will generate same unique id and equals() will return true if literals are same else different literals.

2. Non-Constant pool

- Which is created an object of the string by using new keyword are stored in the non-constant pool memory area
- It allows duplicate object and reference.

1. String behaviours



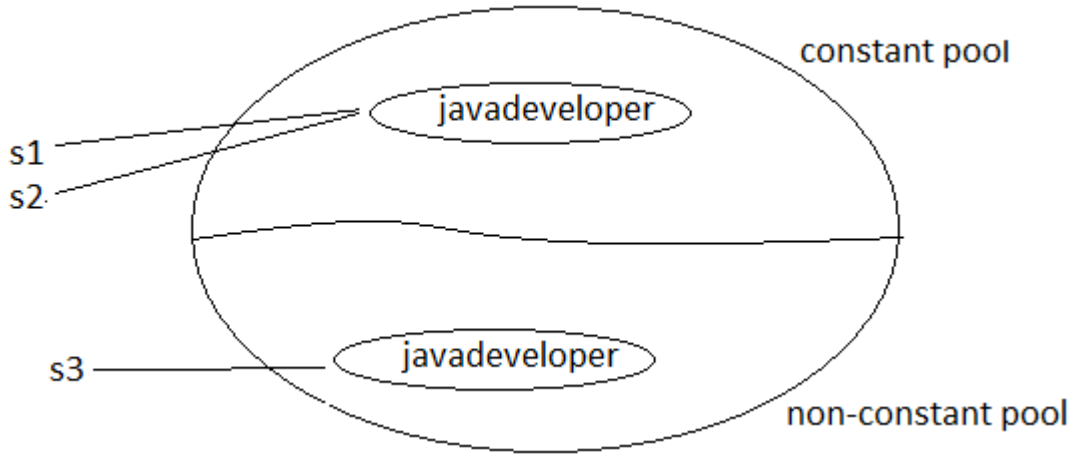
String S1=”developer”;

String s2=new String (“developer”);

S1==s2 →false

S1.equals (s2) →true

2. String behaviour



String s1=”javadeveloper”;

String s2=”java”+”developer”;

String s3=new String (“javadeveloper”);

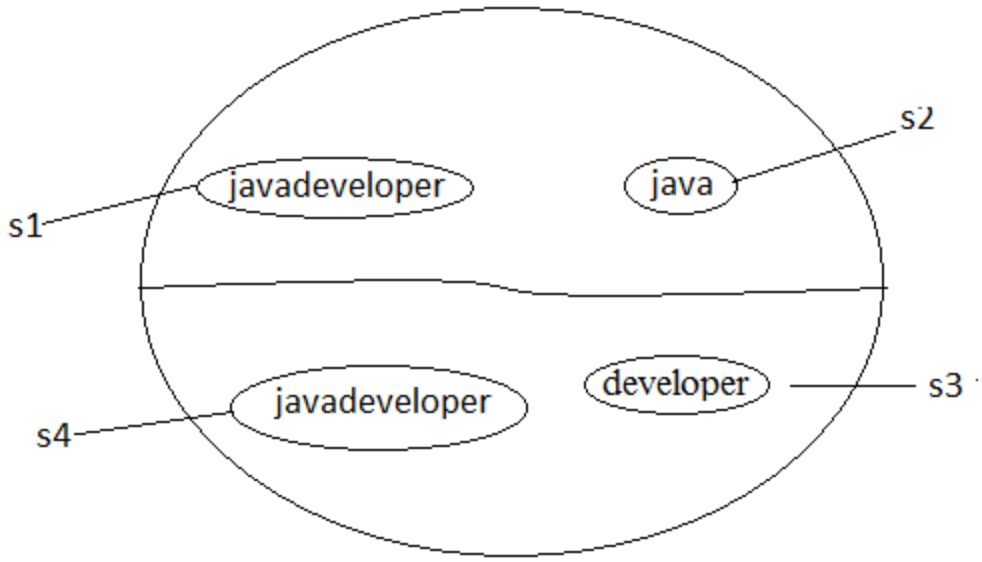
s1==s2 →true

s1.equals (s2) →true

s1==s3 →false

s1.equals (s3) →true

3. String behaviour



String s1=”javadeveloper”;

String s2=”java”;

String s3=s2+”developer”;

String s4=new String (“javadeveloper”)

s1==s3 →false

s1.equals (s3)→true

s3==s4 →false

s3.equals (s4) →true

Note:

- ✓ In a string declare if it joining with reference variable it will be created an object in non-static pool.

Singleton class:

- Allows only one object creation in its class.
- Private constructor allows access outside class members.
- getInstance () returns an object of its own class.

Arrays:

Arrays contain homogeneous data type and its performance is fast.

Array declaration

arratype [] arrayname;

e.g.:

int [] arr1;

double arr2;

Array initialization

arrayname=new arraytype [size];

e.g.:

arr1=new int[5];

arr2=new double [4];

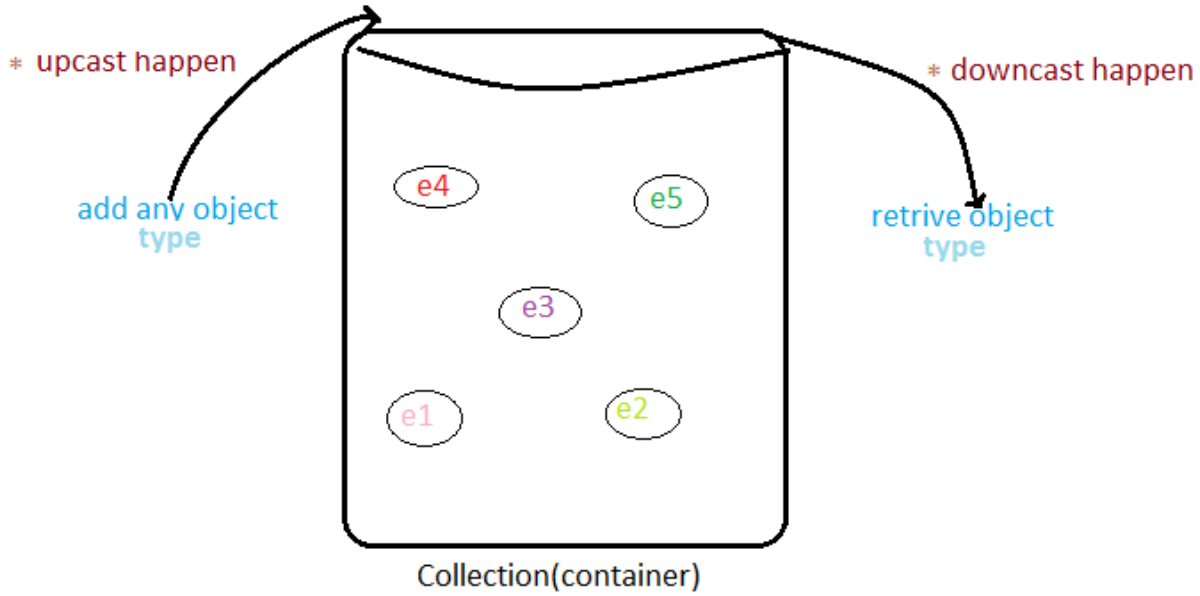
Array Disadvantages:

- Limited size.
- Allow Homogeneous (similar) data type only.
- Length variable(capacity) ;doesn't show the filled array
- Don't follow the data structure principles.

Data structure/Collection API/Frameworks:

- Underlying data structure implementation.
- Resizable/Growable data at runtime.
- Predefined methods for operation.
- Hold either homogeneous or heterogeneous data type.
- Performance is slow.

Fundamentals of collections:



The above figure depicts the behaviour of the collections, the container is a collection which store class objects and interfaces. While adding an element in a collection use add method, we can add any type of object but that should be upcasted to **Object** class type. While retrieving object it must downcast because objects are still in **Object** class type. To retrieve an all object in 2 manner either use foreach loop or iterator interface.

1. Example of foreach loop:

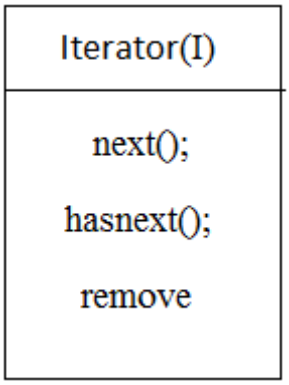
```
for (Object o1: c1)
    {
        system.out.println (o1);
    }
```

2. Example of iterator:

```
Iterator i1=c1.iteraor ();
while (i1.hasnext ())
{
    i1.next ();
}
```

Note:

- ✓ Iterator type object has 3 main methods which is helps to trace out the elements of the collections.
- ✓ **Iterator** is an interface which consist of these methods as shown below:



- ✓ Next (): Get next element from container (e1…..e5) after is it container empty or not i.e. don't know by this method, if there is no element it occurs "**NoSuchElelment Exception"** to overcome use hasnext ().
- ✓ Hasnext (): returns Boolean value if next element is present returns true else returns false.
- ✓ Remove (): remove an element from container.
- ✓ Collections are implemented by **iterable** interface.

Collections API:

Collections are 3 types:

1. List
2. Set
3. Queue

1. List features:

- List is a type of collection
- List can hold any type of object.
- All objects are stored with index.
- List preserve insertion order.
- List allow duplicate object.
- List allows null insertion object.
- Object can retrieve by its index.

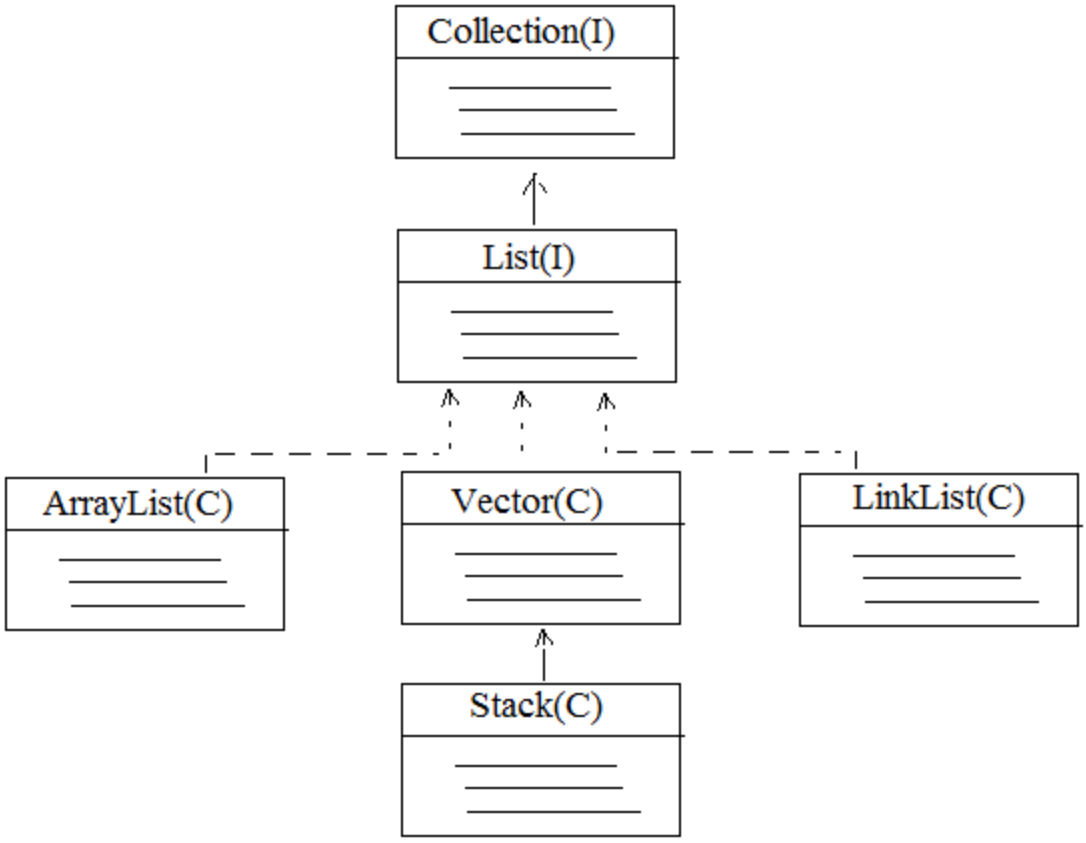


Fig: List class diagram.

1. ArrayList:

- It is a type of serializable.
- It is a type of cloneable.
- It is a random access.
- These are all marker interface means empty interface.
- Its Growable size is **new_capacity= old_capacity*1.5+1.**
- It inherit all the methods from List interface.
- It has 3 type of constructor:
    1. ArrayList (); //default capacity.
    2. ArrayList (capacity);
    3. ArrayList (collection c);

Adv:

- Retrieval by index is fast

Disadv:

- Insertion and deletion to take time.
- We can't see the initial capacity.
- Copying all object to the next interval ArrayList.

2. Vector:

- It inherits the all the methods of List interface.
- It is legacy class in collection from **jdk1.0.**
- All the things are same as ArrayList but vector is **thread safe/thread synchronized.**
- Its Growable size is **new_capacity= old_capacity*2.**
- It has 3 constructor:
  1. Vector (); //default capacity.
  2. Vector(initial capacity);
  3. Vector(collection c);

Adv:

- Retrieval is good when capacity is fixed.
- We can see the initial capacity.
- It is thread safe/synchronized.

Disadv:

- Copying all object to the next interval vector.
- Insert and deletion is to take more time.

3. LinkedList:

- It is a type of serializable and cloneable.
- Implemented with doubly linked list data structure manner.
- An object of linkedlist consist 3 information
  Previous object address
  Object value
  Next object address.
- First object has previous address is **null**, last object next address is **null.**
- It has another iterator to access object using ListIterator, it trace both bi & unidirectional means forward and reverse.

Adv:

- insertion and deletion is good
- It is a type queue features.
- It can traverse both forward and reverse.

Disadv:

- It is not a type of random access.
- It is difficult to retrieval, means slow.

Set collection:

- It is a type of collection.
- Elements are added without index.
- Doesn't preserve insertion order.

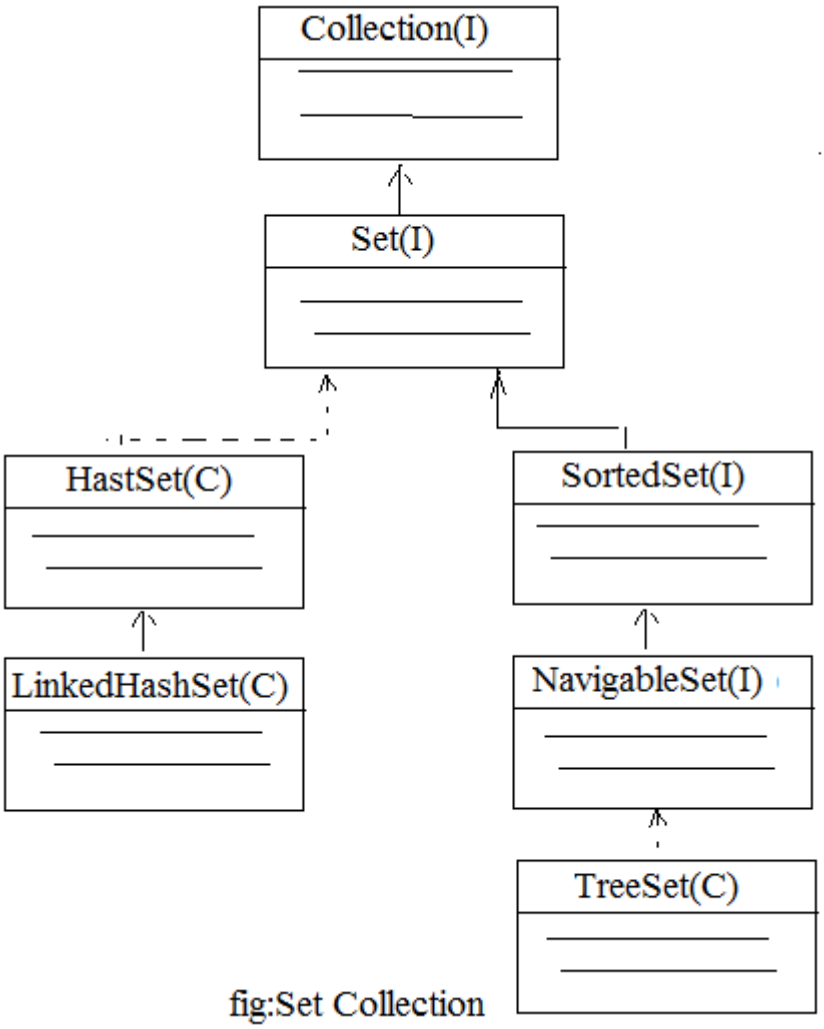


fig:Set Collection

- Duplicates are not allowed but Null is allowed.

HashSet:

- Type of set collection.
- Type of serializable and cloneable.
- Do not preserve insertion order.
- Hashset implemented using hash table data structure.
- Add only unique objects based on hash code.
- Duplicates are not allowed.
- Null insertion allowed.
- When duplicates added it ignores and size is also increase but doesn't show any error.
- It has 4 overloaded constructor:
  1. HashSet s1=new HashSet();//default capacity is 16
  2. HashSet s2=new HashSet(int initial capacity);
  3. HashSet s3=new HashSet(int initial capacity, float fill ration);//default fill ratio/load factor is 0.75
  4. HashSet s4=new HashSet(Collection c);

LinkedHashSet:

- Type of set collection and sub class of hashset.
- Linked hashset implemented using hybrid data structure (combination of different data structure).
- Preserve insertion order.

TreeSet:

- It is a type of Set, SortedSet, and NavigableSet collection.
- Duplicates objects are not allowed.
- Stores only homogeneous objects.
- Sorting:
  Comparison.
  Objects should be same type as first object.
- Objects has to be type comparable.
- Comparable is an inherited from **java.lang** package.
- Comparator is inherited from **java.util** package.
- By default natural sort order.

- Null is not allowed.

Queue:

- It is a type of collection.
- Add any object into queue without index.
- Doesn't preserve insertion order.
- Duplicate objects are allowed.
- Null insertion is not allowed.
- It follows **F**irst **I**n **F**irst **O**ut.

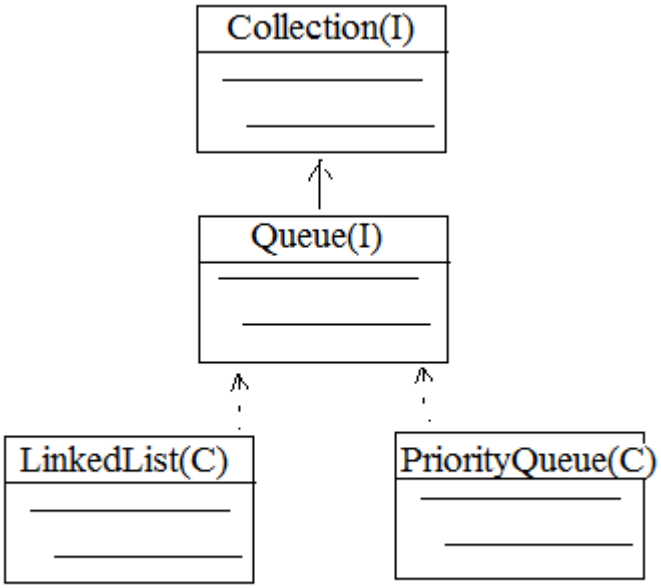


fig.Queue Collection

Priority Queue:

- Internally sorted queue.
- Head element→point's lowest element.
- Trail element→points highest element.
- poll ()→retrieves and remove head element of queue.
- peek ()→retrieve head element only.
- Queue allows comparator object.

Map:

- It keep mapped value (like index).
- Map put object in key=value pair:
  Key is object type

Value is object value.

- Key is should be unique.
- Null allowed to key and value.

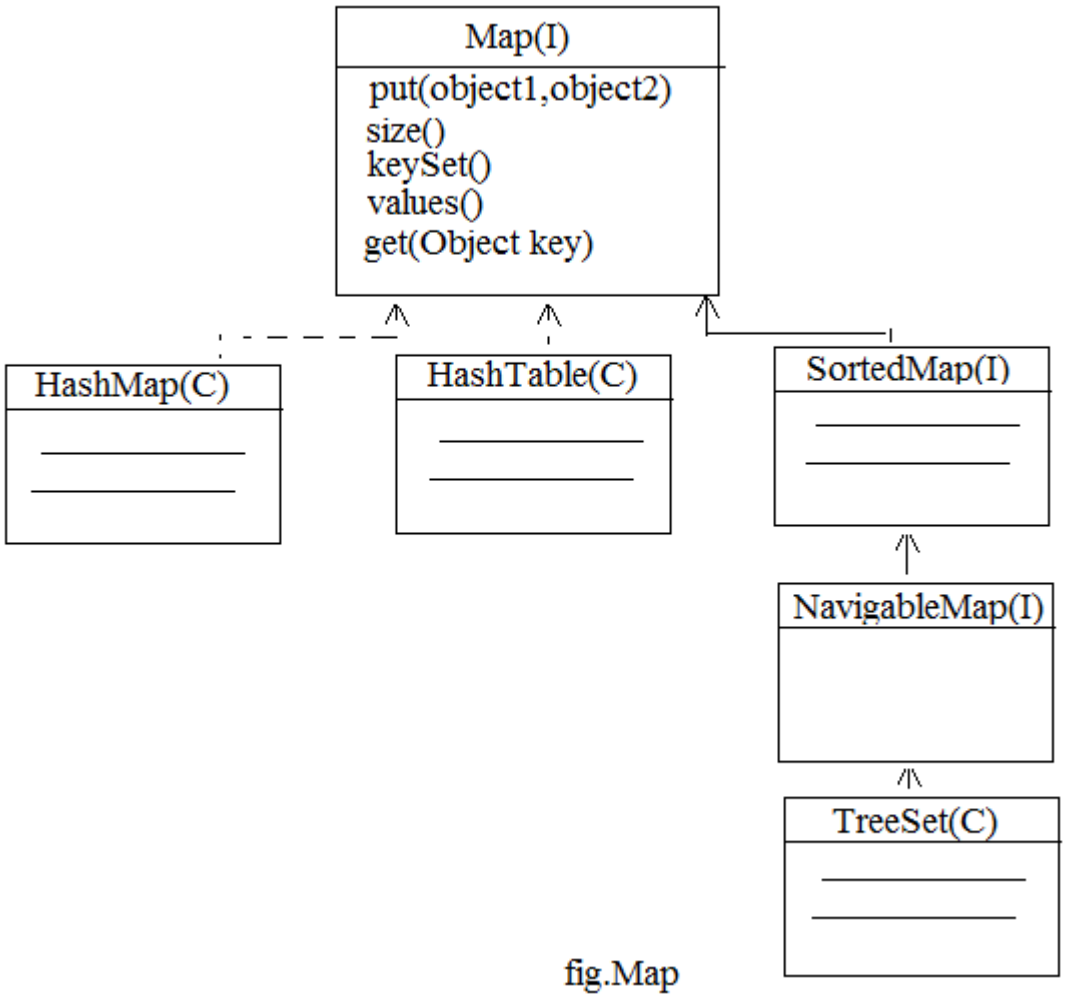


fig.Map

HastTable:

- Null key and value are not allowed.
- Thread safe class.
- Performance is slow because of thread safe.

Exception handling:

Program execution can terminate in 3 manner:

1. Normal termination.
2. Force termination.
3. Abrupt termination.

- Normal termination is terminated after successfully execution of program.
- Force termination is terminated by user eventually.
- Abrupt termination is terminated by JVM, which is a piece of code is not able to execute by JVM. JVM/Run time environment look for code which is capable of handling the execution of the block code is known as **exception handler**.
    - Statement which is not executed by JVM means it is exception case.
    - When exception occur stack and hep memory will be cleared.
    - Compiler will never give an error message for exception.
    - The execution are handled by developing **try/catch** block.
    - The try block should have errors that might generate exception and catch is one which should handle the exception.
    - Try block should be always followed by **catch** block or **finally** block. A try block can have multiple catch block.
    - The catch block should take an argument of type exception type thrown at runtime environment. If an exception occur in try block that will handover to catch block after solving exception it continues the flow of program execution, JVM never go back to try block.
    - The finally block always extended by JVM, the statement written in finally block will be executed even if an unexpected exception occurs. The finally block is very useful when mandatory statements has to be executed by JVM. If user terminate program by System.exit () then finally method will not executed by JVM.
    - **Throw** keyword is used to throw an exception object explicitly in the code, the throw keyword can throw any **throwalbe** type object, throw keyword can be used anywhere in program.
    - The exception occurred in a method body can be delegated its caller explicitly by using **throws** keyword, its declaration keyword should be used in the method or constructor declaration only.

Types of exception:

1. Checked exception
2. Unchecked exception

1. An exception checked by compiler at compile time is known as checked exception, the try/catch block should be written before compilation.
2. Exception which are not checked by compiler and known only at runtime is known as unchecked exception.

Note: user can also develop his own exception.

Threads:

- It is an execution of instance by sharing system resources to do particular task.
- Allocation of system resource decided by JVM.
- Separate stack area is created for every thread, all threads or tasks can run parallel.
- Main method is one of the thread it must execute to run main method in java.
- Thread can be implemented in 2 ways:
    1. By implementing **Runnable** interface. // this is better.
    2. By extending **thread** class.

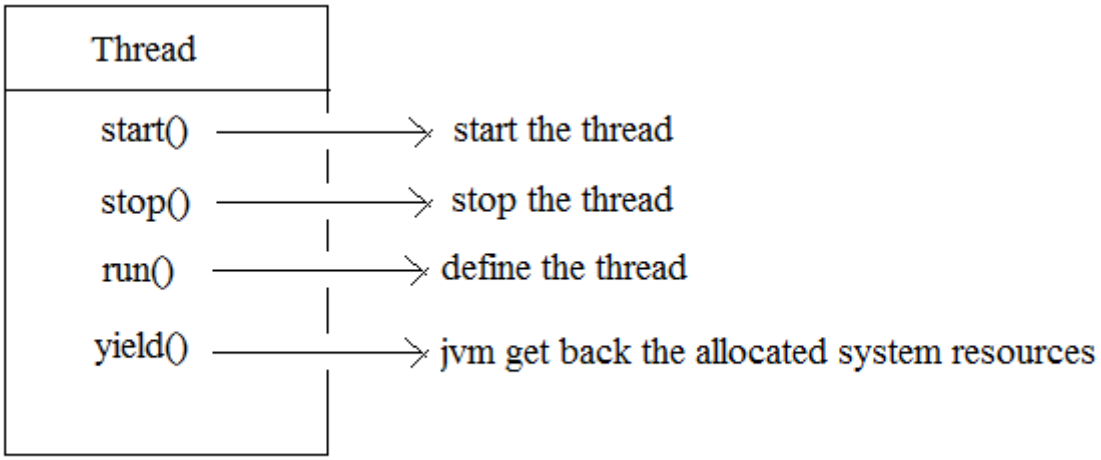


fig. Thread

- Thread has 3 properties:
    1. Thread ID:-Assign by JVM, user can't change it.
    2. Thread name:-thread name can modify .
    3. Thread priority:-Resource allocated based on this property. 1-10 is the priority, 10 is highest and 1 is lowest priority. Default is 5.
- Objects are shared across various threads.

Thread safe/thread synchronized:

When two or more threads need to access shared resources, they need some way to ensure that the resource will be used by only one thread at a time this process is called **synchronization.**

Object lock:

happens in thread synchronization to lock (wait) remain threads to complete execution of current thread, this is achieved by synchronized instance method.

Class lock: achieved by synchronized static method.

Inter thread communication (ITC):

Java includes an inter-process communication mechanism via the **wait ( )**, **notify ( )**, and **notifyAll ( )** methods. These methods are implemented as **final** methods in **Object**, so all classes have them. All three methods can be called only from within a **synchronized** context.

- **wait ( ):** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify( )**.
- **notify( ) :** wakes up a thread that called **wait( )** on the same object.
- **notifyAll( ):** wakes up all the threads that called **wait( )** on the same object. One of the threads will be granted access.

Deadlock:

- A special type of error that you need to avoid that relates specifically to multitasking is ***deadlock,*** which occurs when two threads have a circular dependency on a pair of synchronized objects.
- Happens when a thread is incomplete to run but another thread is waiting for execution then JVM never release until thread completes its task.
  To overcome this we used ITC (inter thread communication) concept.

Thread life cycle:

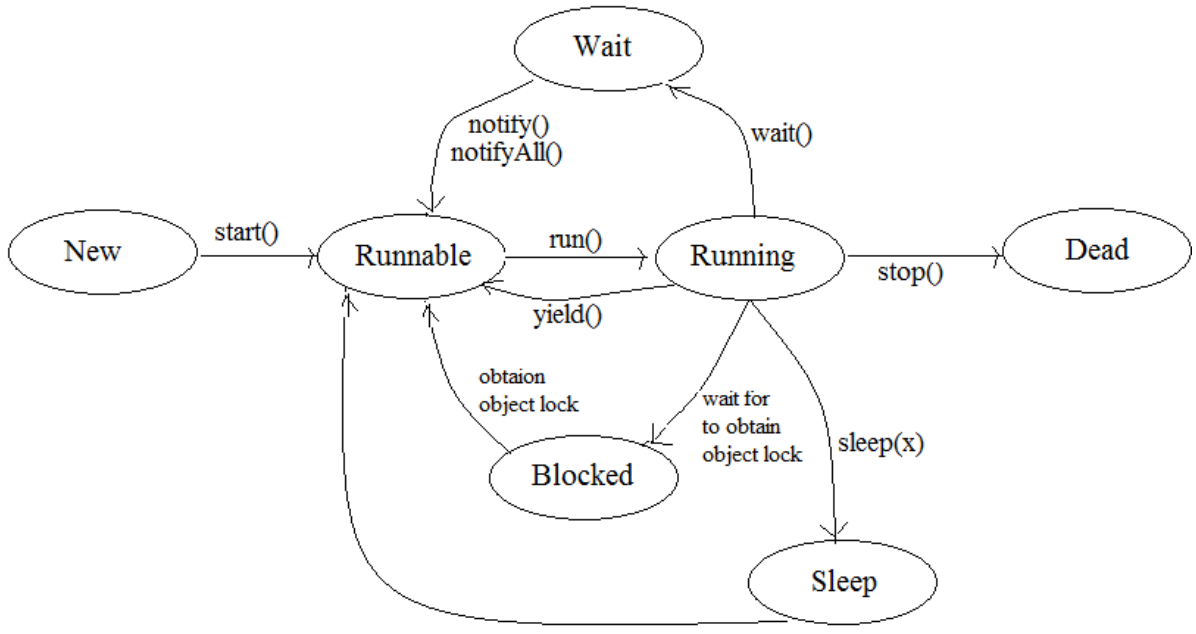


Fig: thread life cycle.

File Handling:

- Defined in java.io package have classes and interfaces.
- To create file/directory.
- To read/write directory.
- **File** defines many methods to obtain the standard properties of a **File** object

File class used to create file directory.
FileWriter class used to write operation.
FileReader class used to read only one character.
BufferReader class used to read a whole line in a file.

Note:

- flush() method is used to view data in a file, until to do flush() can't view a data in file .
- close() method is call to save the resources and increase performance, resources will leak until to close the objects. While closing flush () methods run automatically.

Serialization

- Persistent object means object stored in a file or data base permanently is called **serialization** process.
- Non-persistent object means object erase after system termination or garbage collected or dereferenced object.
- **.ser** extension for serializable object.
- **Transient** keyword used for only variables to not making serializable and static variables also can't make serializable object.
- **Deserialization** means which is retrieving an object from file or database.

Thank you