Chapter 2

- Classes,
- Constructors
- Access Specifiers
- Abstract Classes and Wrapper Classes
- Inheritance, Polymorphism
- Method Overriding
- keyword- Static, final, Super and this
- Garbage collection
- finalize method
- String and mutable string
- Inner Classes

- ***Class-***

  Collection of objects is called class. It is a logical entity.

- ***Object***

  Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

An object has three characteristics:

- **state:** represents the data of an object.
- **behaviour:**represents the behavior of an object.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user, but is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behaviour.

A class in java can contain:

- data member
- method
- constructor
- Block

Syntax to declare a class:

```
class <class name>{
data member;
  method;
}
```

- Simple Example of Object and Class

  In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
class Student{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)

 public static void main(String args[]){
  Student s1=new Student();//creating  an object of Student
  System.out.println(s1.id+" "+s1.name);

 }
}
```

```
Output:0 null
```

Instance variable

- 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.

Method

- In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

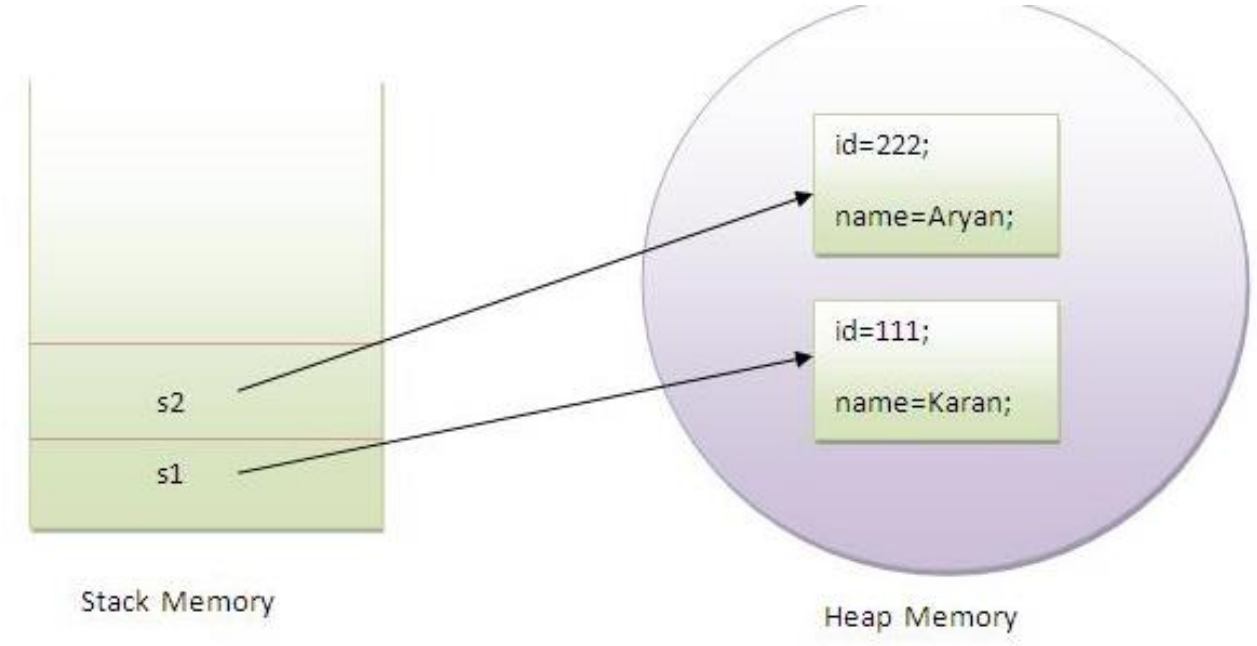
new keyword

- The new keyword is used to allocate memory at runtime.

Example of Object and class that maintains the records of students

```
class Student{
 int rollno;
 String name;

 void insertRecord(int r, String n){  //method
  rollno=r;
  name=n;
 }

 void displayInformation(){System.out.println(rollno+" "+name);}//method

 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();

  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");

  s1.displayInformation();
  s2.displayInformation();

 }
}
```

```
Output:111 Karan
       222 Aryan
```



object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
  r2.calculateArea();
}
}
```

```
Output:55
       45
```

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

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

Annonymous object

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

```
class Calculation{

 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}

public static void main(String  args[]){
 new Calculation().fact(5);//calling  method with annonymous object
}
}
```

```
Output:Factorial is 120
```

Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
  r2.calculateArea();
}
}
```

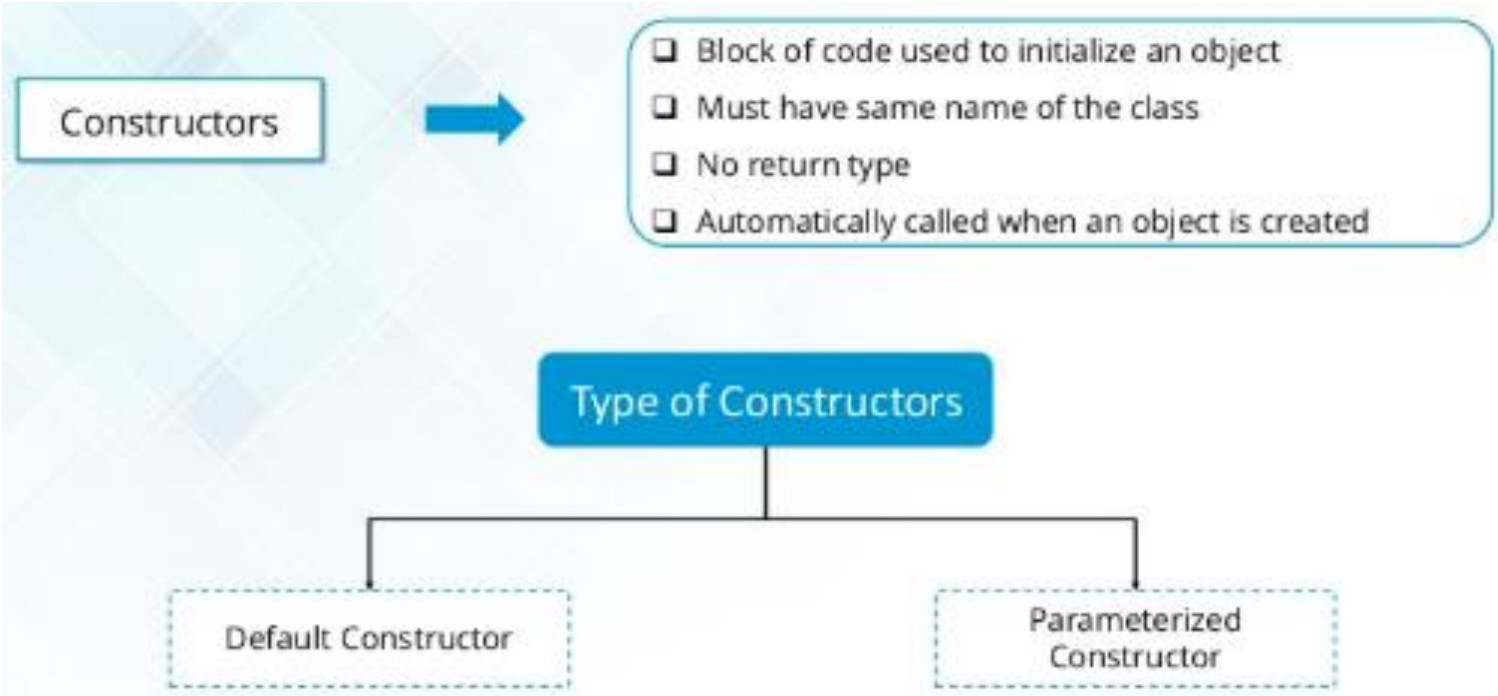
```
Output:55
       45
```

- **Constructor** is a **special type of method** that is used to initialize the object.
- Constructor is **invoked at the time of object creation**. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating constructor:

- Constructor name must be same as its class name
- Constructor must have no explicit return type

constructor



- **Default Constructor**
- A constructor that have no parameter is known as default constructor.
- Syntax of default constructor:
- <class name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike{

Bike(){System.out.println("Bike  is created");}

public static void main(String  args[]){
Bike b=new Bike();
}
}
```

```
Output: Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

- Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
class Student{
int id;
String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student s1=new Student();
Student s2=new Student();
s1.display();
s2.display();
}
}
```

```
Output:0 null
       0 null
```

Parameterized constructor

- A constructor that have parameters is known as parameterized constructor.
- Parameterized constructor is used to provide different values to the distinct objects.

```
class Student{
   int id;
   String name;

   Student(int i,String n){
   id = i;
   name = n;
   }
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   s1.display();
   s2.display();
  }
}
```

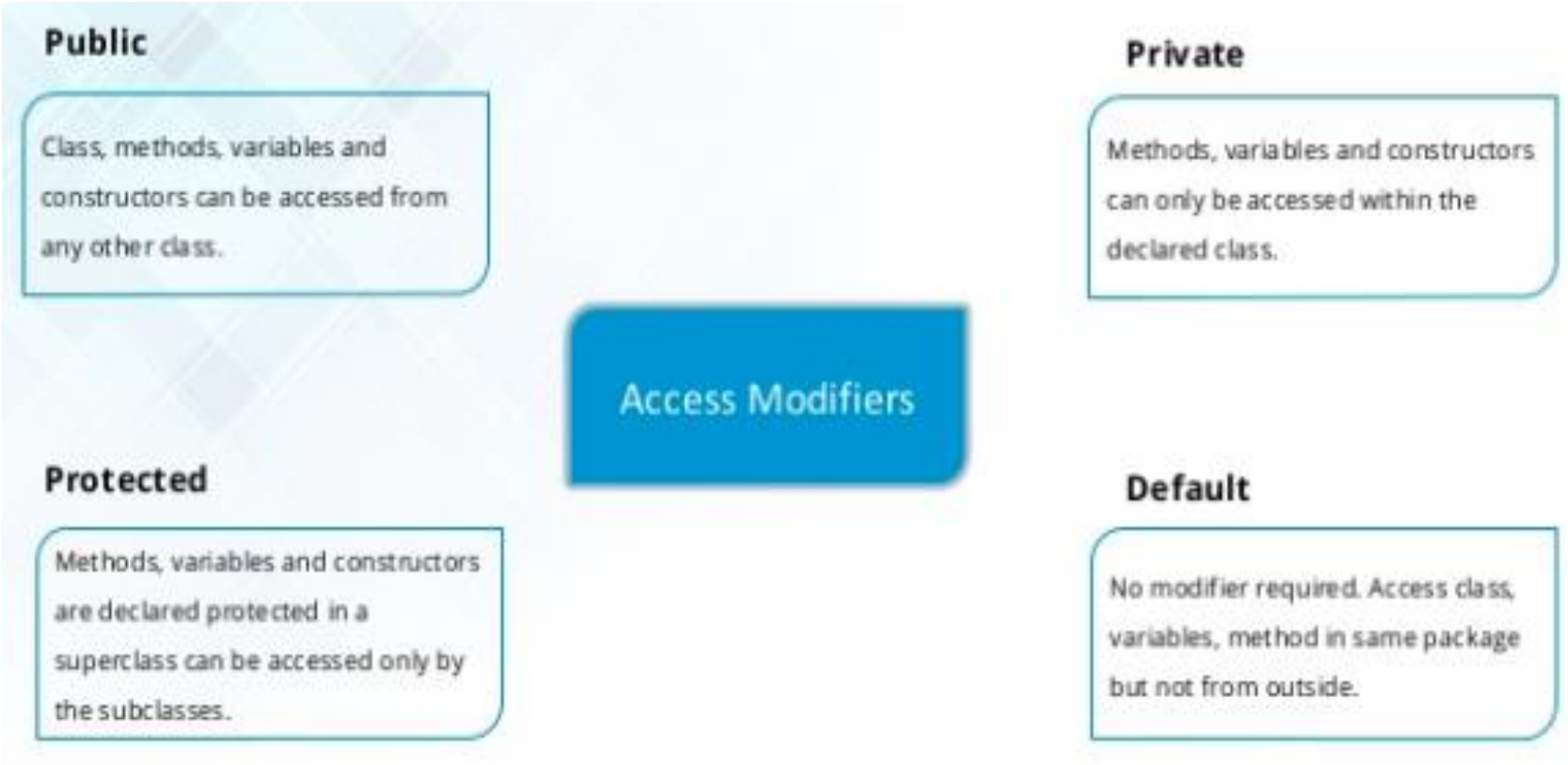
```
Output:111 Karan
       222 Aryan
```

Constructor Overloading

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

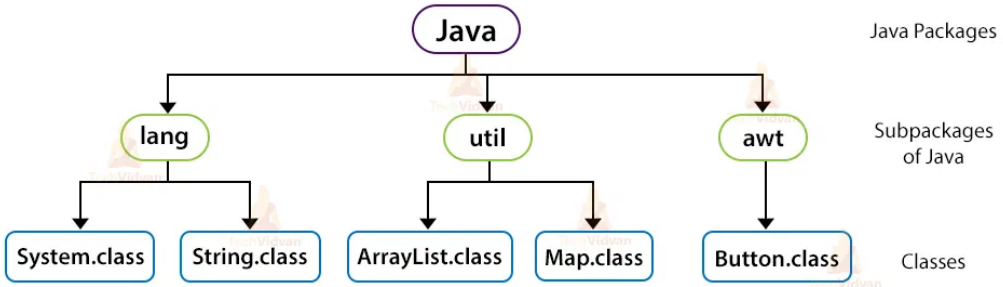
```
class Student{
   int id;
   String name;
   int age;
   Student(int i,String n){
   id = i;
   name = n;
   }
   Student(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
   void display(){System.out.println(id+" "+name+" "+age);}

   public static void main(String args[]){
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Access Specifiers



Access Modifiers	Same Class	Same Package	Sub Class	Other Packages
Public	Y	Y	Y	Y
Private	Y	N	N	N
Protected	Y	Y	Y	N
Default	Y	Y	N	N

Built-in Packages in Java



- **Abstract class:**
- A class that is declared as abstract is known as **abstract class**.
- It needs to be extended and its abstract method implemented.
- It cannot be instantiated.
- It can have non abstract methods too.
- **Syntax to declare the abstract class**
- **abstract class** <class_name>{}
- **abstract method**
- A method that is declared as abstract and does not have implementation is known as abstract method.
- **Syntax to define the abstract method**
- **abstract** return type <method name>();//no braces{}

Example of abstract class and abstract method

```
abstract class Shape
{
abstract void draw();
}

class Rectangle extends Shape
{
void draw(){System.out.println("drawing rectangle");}
}

class Circle extends Shape
{
void draw(){System.out.println("drawing circle");}
}

class Test{
public static void main(String args[])
{
Shape S1=new Circle();
S1.draw();

Shape S2=new Rectangle();
S2.draw();


}
}
```

```
Output:
drawing circle
drawingRectangle
```

Rule1 : If there is any abstract method in a class, that class must be abstract.

Rule2 : If you are extending any abstact class that have abstract method, you must either provide the implementation of the method or make this class abstract.

Interface in Java

Interfaces

- Interface in java is a blueprint of a class.
- Each method in an interface are implicitly public and abstract.
- It does not contain any constructors.

Example :

```
interface Car {

   void changeGear (int newValue);

   void speedup (int increment);

   void applyBrakes (int decrement);
}
```

- An interface is a blueprint of a class. It has static constants and abstract methods.
- The interface is a mechanism to achieve fully abstraction in java.
- There can be only abstract methods in the interface.
- It cannot be instantiated just like abstract class.

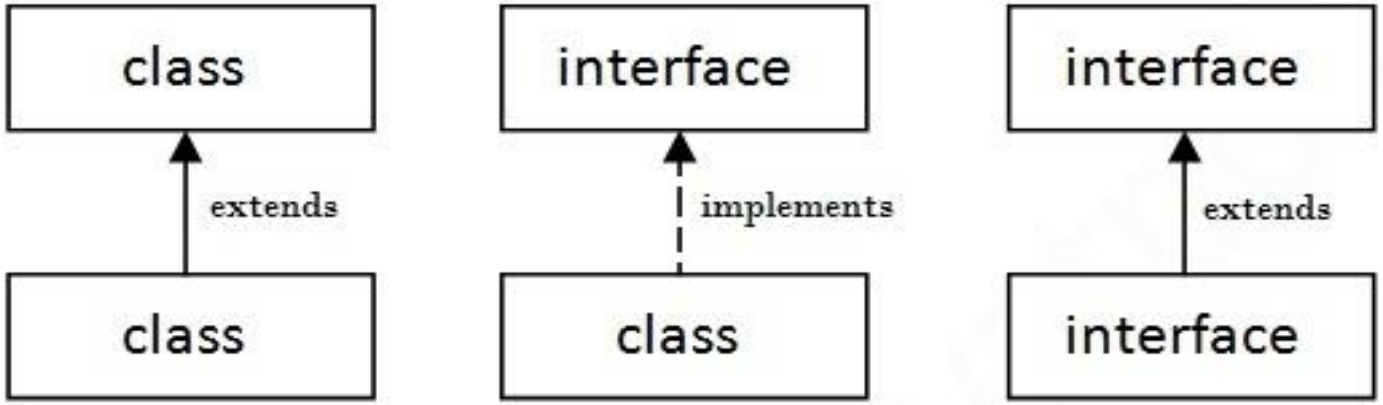
Why use Interface?

There are mainly three reasons to use interface.

- 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.

  Interface fields are public, static and final by default, and methods are public and abstract.

Understanding relationship between classes and interfaces



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

class A implements printable
{
public void print()
{System.out.println("Hello");}

public static void main(String args[])
{
A obj = new A();
obj.print();
 }
}
```

```
Output:  Hello
```

Inheritance in Java

- **Inheritance** is a mechanism in which one object acquires all the properties and behaviors of parent object.
- Inheritance represents the **IS-A relationship**

Why use Inheritance?

- For Method Overriding (So Runtime Polymorphism).
- For Code Reusability.

Syntax of Inheritance

class Subclass-name **extends** Super class-name

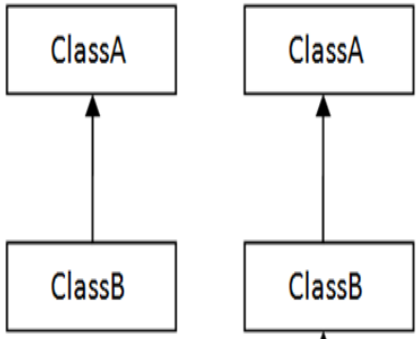
```
{
  //methods and fields
}
```

Example on Inheritance

```
class Employee{
 float salary=40000;
}

class Programmer extends Employee{
 int bonus=10000;

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

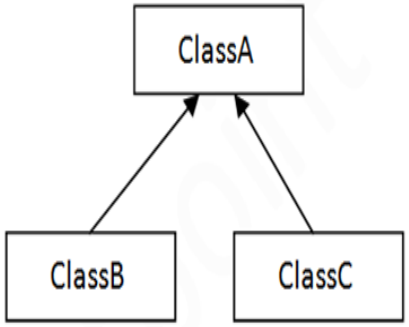
```
Output: Programmer salary is:40000.0
        Bonus of programmer is:10000
```

Single, multilevel and hierarchical possible to implement



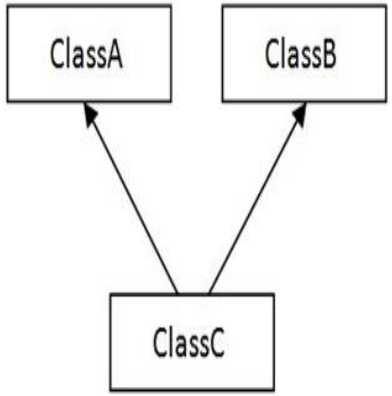
1) Single

2) Multilevel

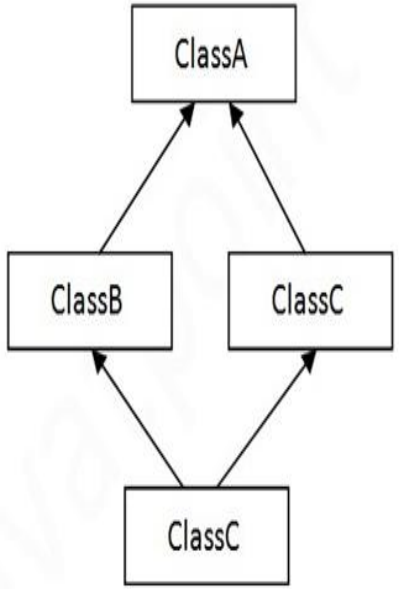


3) Hierarchical

Multiple and Hybrid is supported through interface only



4) Multiple



5) Hybrid

Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

Multiple Inheritance in Java

Programs on Interface-Example

- simple interface.txt
- nested interface.txt
- interface_inside_interface.txt
- interface wih function.txt
- abstract class with interface.txt

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
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 static methods, main method or 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();
	}

Method Overriding in Java

- If subclass provides the specific implementation of the method i.e. already provided by its parent class, it is known as Method Overriding**.**

Advantage of Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding:

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be inheritance (IS-A) relationship.

Example –Method overriding

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike obj = new Bike();
obj.run();
}
```

```
Output:Bike is running safely
```

static keyword

- The **static keyword** is used in java mainly for memory management.
- We may apply static keyword with variables, methods, blocks and nested class.

static variable

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.
- It makes your program **memory efficient**

```
class Counter{
static int count=0;//will get memory only once and retain its value

Counter(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();

}}
```

```
Output:1
       2
       3
```

Static method

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

```
//Program to get cube of a given number by static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

```
Output:125
```

There are two main restrictions for the static method they are:

- The static method can not use non static data member or call non-static method directly.
- this and super cannot be used in static context.

this keyword:

Here is given the 6 usage of 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.

```java
class student{
  int id;
  String name;

  student(int id,String name){
  id = id;
  name = name;
  }
  void display(){System.out.println(id+" "+name);}

  public static void main(String args[]){
  student s1 = new student(111,"Karan");
  student s2 = new student(321,"Aryan");
  s1.display();
  s2.display();
  }
}
```

```
Output:0 null
       0 null
```

In the above example, parameter (formal arguments) and instance variables are same

```java
//example of this keyword
class Student{
  int id;
  String name;

  student(int id,String name){
  this.id = id;
  this.name = name;
  }
  void display(){System.out.println(id+" "+name);}
  public static void main(String args[]){
  Student s1 = new Student(111,"Karan");
  Student s2 = new Student(222,"Aryan");
  s1.display();
  s2.display();
}
}
```

```
Output111 Karan
       222 Aryan
```

- quiz1.txt
- quiz2.txt

super keyword

- The **super** is a reference variable that is used to refer immediate parent class object**.**
- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of super Keyword

- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

super is used to refer immediate parent class instance variable

```
//example of super keyword

class Vehicle{
  int speed=50;
}

class Bike extends Vehicle{
  int speed=100;

  void display(){
    System.out.println(super.speed);//will print speed of Vehicle now
  }
  public static void main(String args[]){
    Bike b=new Bike();
    b.display();

  }
}
```

super() is used to invoke immediate parent class constructor

```
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
}

class Bike extends Vehicle{
  Bike(){
    super();//will invoke parent class constructor
    System.out.println("Bike is created");
    }
  public static void main(String args[]){
    Bike b=new Bike();

}
}
```

```
Output: Vehicle is created
        Bike is created
```

super() is added in each class constructor automatically by compiler.

super is used to invoke immediate parent class method

```
class Person{
void message(){System.out.println("welcome");}
}

class Student extends Person{
void message(){System.out.println("welcome to java");}

void display(){
message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}

public static void main(String args[]){
Student s=new Student();
s.display();
}
}
```

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1.Variable

2.method

3. class

Java final variable

- If you make any variable as final, you cannot change the value of final variable(It will be constant).
- A final variable that have no value it is called blank final variable or uninitialized final variable.
- It can be initialized in the constructor only.

```
class Bike9{
 final int speedlimit=90;//final  variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

```
Output: Compile Time Error
```

Java final method

- If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}
class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

```
Output: Compile Time Error
```

Java final class

- If you make any class as final, you cannot extend it.

```
final class Bike{}
  class Honda1  extends Bike{
  void run(){System.out.println("running  safely with  100kmph");}

  public  static void  main(String  args[]){
  Honda1 honda=  new Honda1();
  honda.run();
  }
}
```

Output: Compile Time Error

Garbage Collection in Java

- In Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.
- Garbage collector is best example of Daemon thread as it is always running in background.
- Main objective of Garbage Collector is to free heap memory by destroying **unreachable objects.**
- **Ways to make an object eligible for GC**
    - Nullifying the reference variable
    - Re-assigning the reference variable
    - Object created inside method
    - Anonymous object

1.By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2.By Re-assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;
```

//now the first object referred by e1 is available for garbage collection

3) Object created inside a method:

- When a method is called it goes inside the stack frame.
- When the method is popped from the stack, all its members dies and if some objects were created inside it then these objects becomes unreachable or anonymous after method execution and thus becomes eligible for garbage collection.

```
staticvoidshow()
   {
     //object t1 inside method becomes unreachable when show()removed
     from stack
Test t1 = newTest("t1");
       display();
 }
```

4) Anonymous object : The reference id of an anonymous object is not stored anywhere. Hence, it becomes unreachable.

```
      new Employee();
```

Ways for requesting JVM to run Garbage Collector

- Once we made object eligible for garbage collection, it may not destroy immediately by garbage collector.
- Whenever JVM runs Garbage Collector program, then only object will be destroyed.
- We can also request JVM to run Garbage Collector.
- Using System.gc() method : System class contain static method gc() for requesting JVM to run Garbage Collector.

Finalization

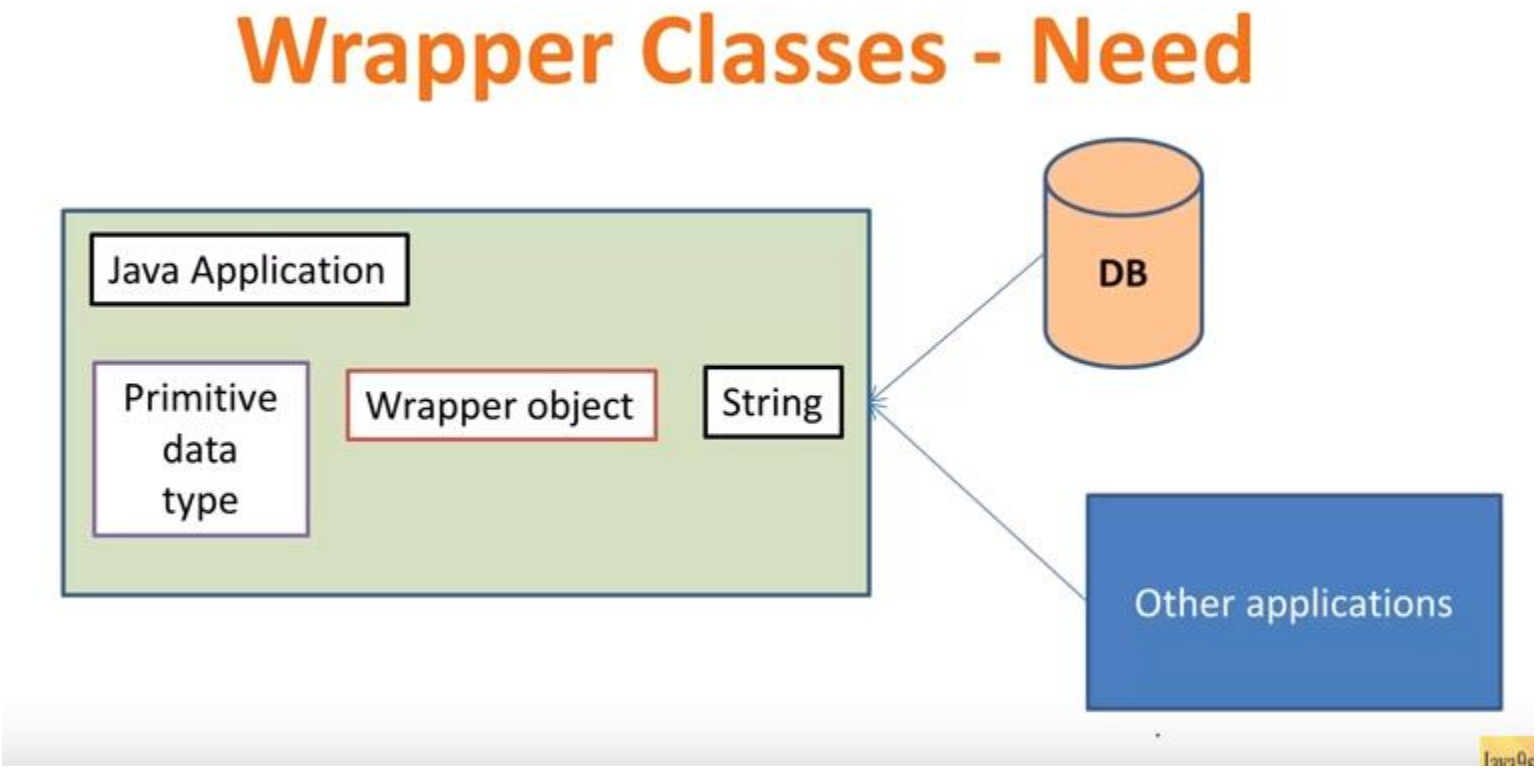
- Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.

finalize() method is present in Object class with following prototype.

- protected void finalize() throws Throwable
- Based on our requirement, we can override *finalize()* method to perform our cleanup activities like closing connection from database.

- gc.txt

Wrapper classes in JAVA



- In development, we come across situations where we need to use objects instead of primitive data types.
- For example, you can’t pass a primitive type by reference to a method.
- Also, many of the standard data structures implemented by Java operate on objects, which mean that you can’t use these data structures to store primitive.
- Wrapper classes encapsulate a primitive type within an object.
- wrapper class.txt

Most of the objects collection store objects and not primitive types.

Primitive types can be used as object when required.

As they are objects, they can be stored in any of the collection and pass this collection as parameters to the methods.

Wrapper classes are classes that allow primitive types to be accessed as objects.

Wrapper class is wrapper around a primitive data type because they "wrap" the primitive data type into an object of that class.

Primitive Data Types and Wrapper Classes

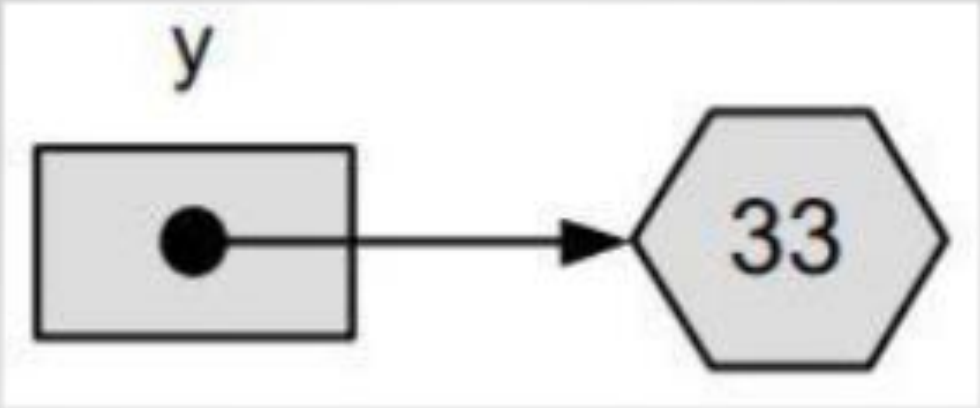
Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Difference b/w Primitive Data Type and Object of a Wrapper Class

The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

```
int x = 25;

Integer y = new Integer(33);
```

The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.



Clearly x and y differ by more than their values:

x is a variable that holds a value;

y is an object variable that holds a reference to an object.

So, the following statement using x and y as declared above is not allowed:

```
int z = x + y;  // wrong!
```

The data field in an Integer object is only accessible using the methods of the Integer class.

One such method is intValue() method which returns an int equal to the value of the object, effectively "unwrapping" the Integer object:

```
int z = x + y.intValue();  // OK!
```

What is the need of Wrapper Classes?

There are three reasons that we might use a Number object rather than a primitive:

As an argument of a method that expects an object (often used when manipulating collections of numbers).

To use constants defined by the class, such as MIN_VALUE and MAX_VALUE, that provide the upper and lower bounds of the data type.

To use class methods for converting values to and from other primitive types, for converting to and from strings, and for converting between number systems (decimal, octal, hexadecimal, binary).

Boxing and Unboxing

- The wrapping is done by the compiler.
- if we use a primitive where an object is expected, the compiler boxes the primitive in its wrapper class.
- Similarly, if we use a number object when a primitive is expected, the compiler un-boxes the object.

Example of boxing and unboxing:

- Integer x, y; x = 12; y = 15; System.out.println(x+y);
- When x and y are assigned integer values, the compiler boxes the integers because x and y are integer objects.
- In the println() statement, x and y are unboxed so that they can be added as integers.

String and mutable string

There are two ways to create String object:

1) String literal

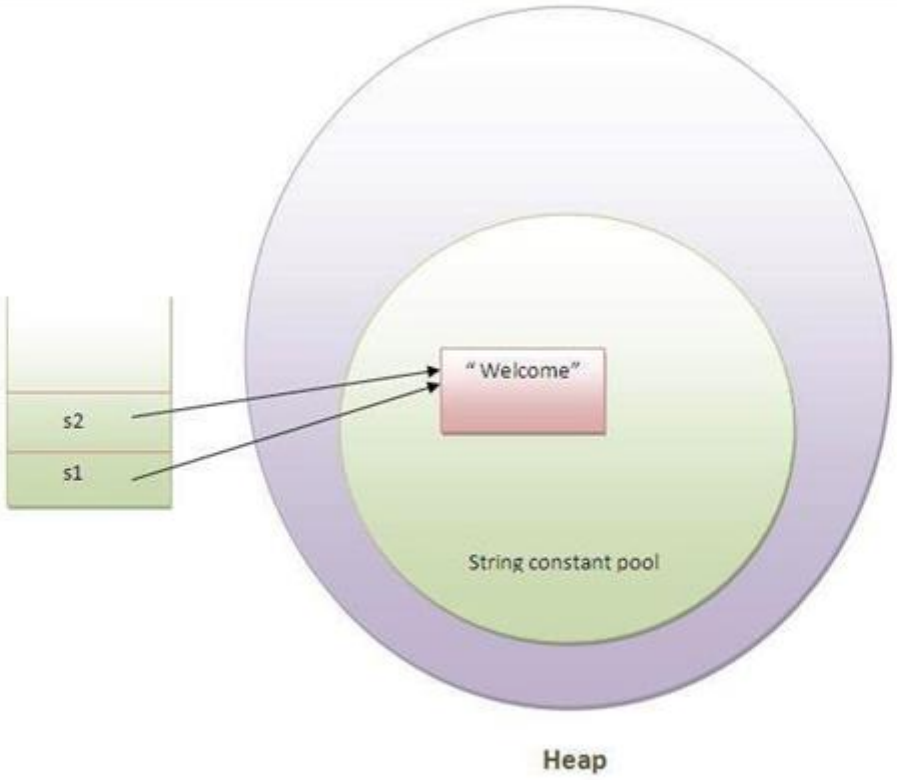
String s="Hello";

- Each time you create a string literal, the JVM checks the string constant pool first.
- If the string already exists in the pool, a reference to the pooled instance returns.
- If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.

For example:

String s1="Welcome";

String s2="Welcome";//no new object will be created



Note: **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

2) By new keyword

- **String s=new String("Welcome");**
  //creates two objects and one reference variable
- In such case, JVM will create a new String object in normal(non pool) Heap memory and the literal "Welcome" will be placed in the string constant pool.
- The variable s will refer to the object in Heap(non pool).

Immutable String in Java

- In java, **string objects are immutable**. Immutable simply means not modifiable or unchangeable.
- Once string object is created its data or state can't be changed but a new string object is created.

```
class Simple{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the e
     nd
   System.out.println(s);//will print Sachin because strings are immutabl
     e objects
 }
}
```

Output:Sachin

Mutable Strings in Java

- In mutable string the value is changed in the address itself
- StringBuffer and StringBuilder classes are used for creating mutable string

Example:

```
class A{
public static void main(String args[]){
StringBuffersb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

There are three ways to **compare String objects:**

- By equals() method
- By = = operator
- By compareTo() method

There are two ways to **concat string objects:**

- By + (string concatenation) operator
- By concat() method

Substring in Java

- **public String substring(intstartIndex):**
- This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- **public String substring(intstartIndex,intendIndex):**
- This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

Nested classes in Java

- Nested classes are used to logically group classes in one place so that it can be more readable and maintainable code
- It can access all the members of outer class including private members

Syntax of Nested class

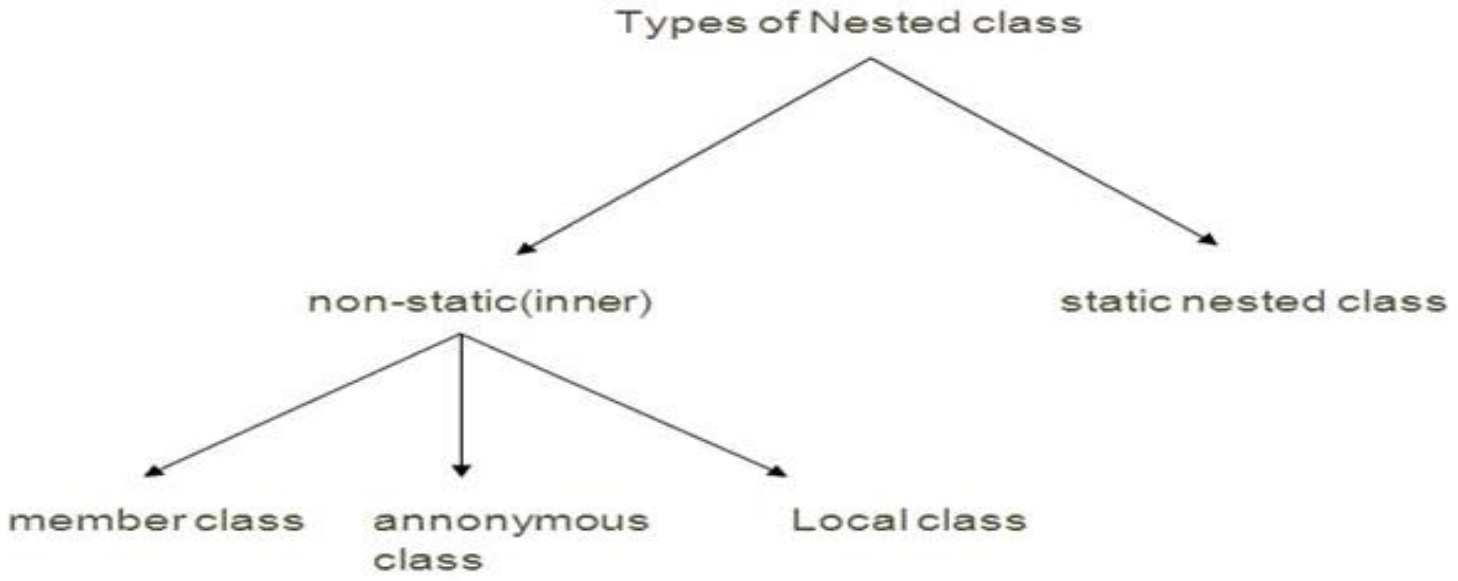
```
class Outer_class_Name{
    ...
 class Nested_class_Name{
  ...
\ }
 ...
}
```

Advantage of nested classes

- 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.
- Nested classes can lead to more readable and maintainable code because it logically group classes in one place only.
- Code Optimization as we need less code to write.

Types of Nested class:

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by compiler.
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

- A class that is declared inside a class but outside a method is known as member inner class.
- **Invocation of Member Inner class**
- From within the class
- From outside the class
- **inner class1.txt**

2. Anonymous inner class

- A class that have no name is known as anonymous inner class. Anonymous class can be created by:
- Class (may be abstract class also).
- Interface
- inner class2.txt

3.Local inner class

- A class that is created inside a method is known as local inner class. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

localInner.txt

- **Rules for Local Inner class**

1.Local variable can't be private, public or protected.

2. Local inner class cannot be invoked from outside the method.

3. Local inner class cannot access non-final local variable**.**

4.Static nested class

- A static class that is created inside a class is known as static nested class. It cannot access the non-static members.
- It can access static data members of outer class including private.
- static nested class cannot access non-static (instance) data member or method.

staticNestedClass.txt