Wrapper classes in Java

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*
- **autoboxing** and **unboxing** fe ature converts primitives into objects and objects into primitives automatically.

- The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

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

Wrapper class Example: Primitive to Wrapper

```
1.//Java program to convert primitive into objects
2.//Autoboxing example of int to Integer
3.public class WrapperExample1{
4.public static void main(String args[]){
5.//Converting int into Integer
6.int a=20;
7.Integer i=Integer.valueOf(a);//converting int into Integer explicitly
8.Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
9.
10.System.out.println(a+" "+i+" "+j);
11.}}
```

Output:

```
20 20 20
```

Wrapper class Example: Wrapper to Primitive

```
1.//Java program to convert object into primitives
2.//Unboxing example of Integer to int
3.public class WrapperExample2{
4.public static void main(String args[]){
5.//Converting Integer to int
6.Integer a=new Integer(3);
7.int i=a.intValue();//converting Integer to int expli
citly
8.int j=a;//unboxing, now compiler will write a.intV
alue() internally
9.
10.System.out.println(a+" "+i+" "+j);
11.}}
```

Output:

```
3 3 3
```

CONSTRUCTORS IN JAVA

- In Java, a constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Example of default constructor

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

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

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Example of parameterized constructor

```
class Student4{
    int id;
    String name;
     //creating a parameterized construc
tor
    Student4(int i,String n){
    id = i;
    name = n;
    }
//method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

The syntax of Java Inheritance

1.class Subclass-name **extends** Superclass-name
2.{
3. //methods and fields
4.}

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

SINGLE 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.bonu
s);
}
}
```

Method Overloading in Java

Method Overloading: changing no. of arguments

```
class Adder{
 int add(int a,int b){return a+b;}
int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Method Overloading in Java

Method Overloading: changing data type of arguments

```
class Adder{
int add(int a, int b){return a+b;}
double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Is Method Overloading possible by changing the return type of method only?

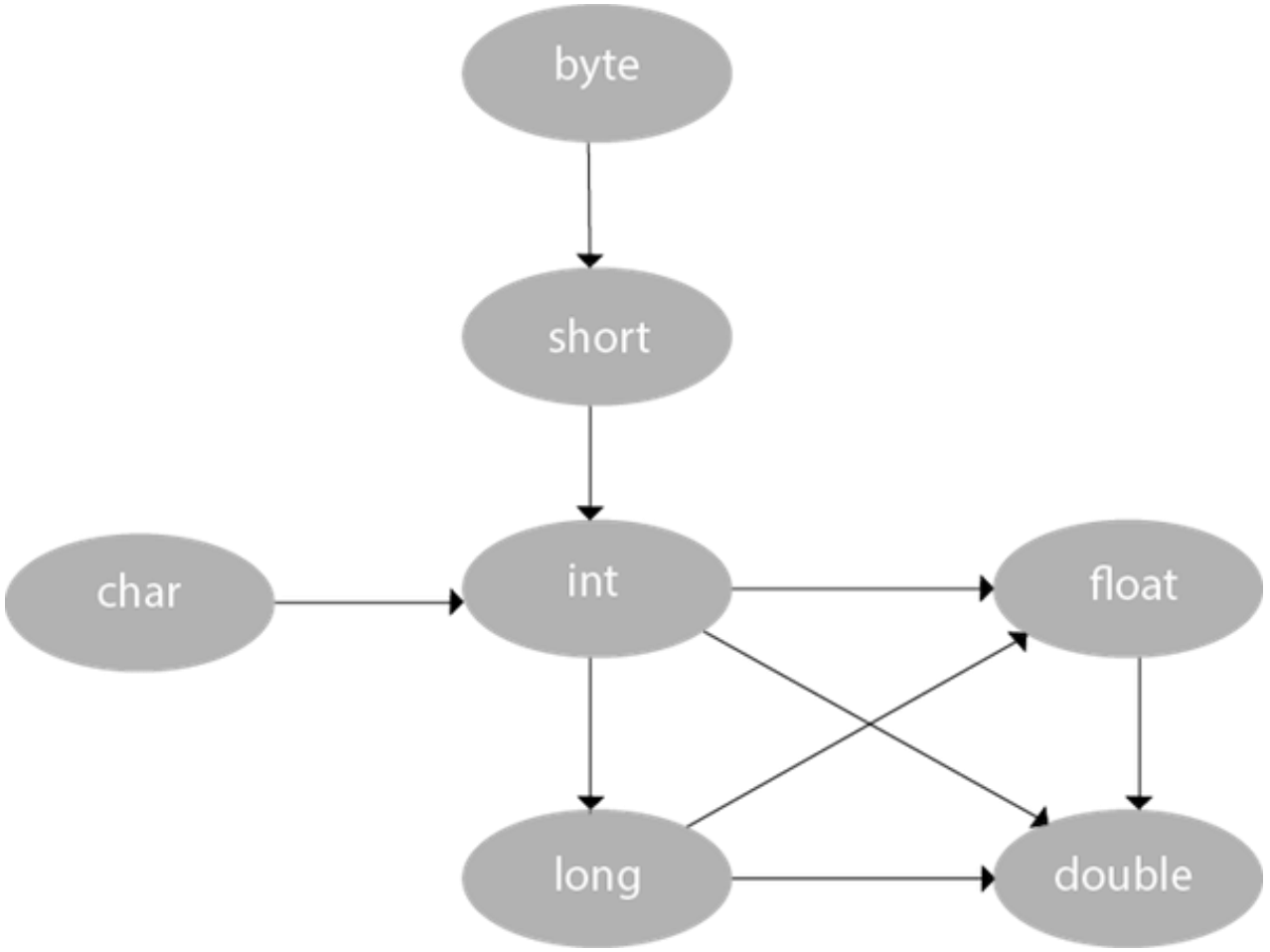
```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
1.class TestOverloading4{
2.public static void main(String[] args){System.out.println("main wit
h String[]");}
3.public static void main(String args){System.out.println("main with
String");}
4.public static void main(){System.out.println("main without args");}

5.}
```

Method Overloading and Type Promotion



```
class OverloadingCalculation3{
  void sum(int a,long b){System.out.println("a method invo
ked");}
  void sum(long a,int b){System.out.println("b method invo
ked");}

  public static void main(String args[]){
   OverloadingCalculation3 obj=new OverloadingCalculatio
n3();
  obj.sum(20,20);//now ambiguity
  }
}
```

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

Rules for Java Method Overriding

1.The method must have the same name as in the parent class

2.The method must have the same parameter as in the parent class.

3.There must be an IS-A relationship (inheritance).

```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Q. Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, AXIS BANK, ICICI and HDFC Bank could provide 5%, 6%, and 10% rate of interest.

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1.super can be used to refer immediate parent class instance variable.

2.super can be used to invoke immediate parent class method.

3.super() can be used to invoke immediate parent class constructor.

super is used to refer immediate parent class instance variable.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

super can be used to invoke parent class method

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

super is used to invoke parent class constructor.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

final keyword

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).

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

 Bike9 obj=new  Bike9();
 obj.run();
 }
```

Java final method

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

```
1.class Bike{
2.  final void run(){System.out.println("running");}
3.}
4.
5.class Honda extends Bike{
6.    void run(){System.out.println("running safely with 100kmp
h");}
7.
8.    public static void main(String args[]){
9.    Honda honda= new Honda();
10.    honda.run();
11.    }
12.}
```

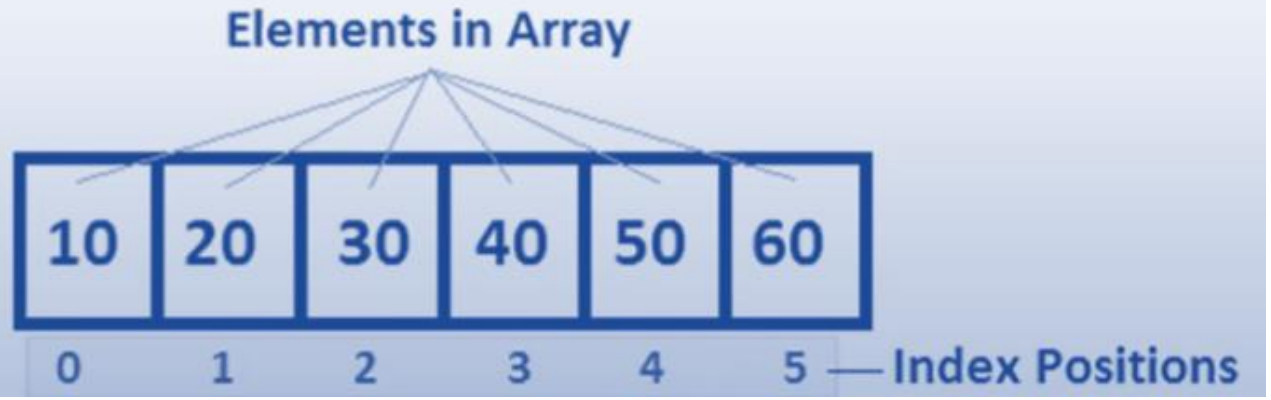
Java final class - If you make any class as final, you cannot extend it.

```
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();
  }
}
```

What is Array ?

- An array is an object that holds a fixed number of values of homogeneous or similar data-type.
- Or say An Array is a Data Structure where we store similar elements.
- The length of an array is assigned when the array is created and After creation, its length is fixed.
- For example : int a[ ]=new int[6];

  It will create an array of length 6 and index value will always start from 0.



Features Of An Array :

- A Java array variable can be declared like other variables with [ ] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- In Java, Arrays are objects, and thus they occupy memory in 'Heap Area'.
- The direct superclass of an array type is Object.
- They are always created at runtime.
- The length of an array can be find by using member 'length'.
  This is different from C/C++ where we find length using sizeof.
- The elements of array are stored in consecutive memory locations.

Single Dimensional (1-D)

Declaration Of An Array :

- Diferent Ways of Declaration of Arrays are :
  1. int[ ] a;
  2. int [ ]a;
  3. int[ ]a;
  4. int a[ ];

- Most Preferred Array Declaration is ‘int[ ] a;’, because here ‘a’ is one dimensional int array, thus name is clearly separated with type.

- We cannot provide size at the time of array declaration i.e. ‘int[3] a;’ or ‘int a[3];’ statement is incorrect.

- Note that, there is difference between below two statements :
  int[ ] a,b; // here ‘a’ and ‘b’ both are arrays.
  int a[ ], b; // here ‘a’ is array and ‘b’ is simple int type variable, not an array.

Single Dimensional (1-D)

Creation Of An Array :

- We can create an array after declaration as follows :

```
int[ ] a;  //array declaration
a=new int[3]; // array creation
```

- It is compulsory to declare the size of an array at the time of creation.

- We can declare and create array within a single line as follows :

```
int[ ] a=new int[3];
```

- If we declare size of an array as '0' i.e. 'int[ ] a=new int[0];', then program will successfully compile and execute.

- If we declare size of an array as negative i.e. 'int[ ] a=new int[-3];', then the program will compile successfully but when we run the program, it will throw 'NegativeArraySizeException' exception.

Single Dimensional (1-D)

Retrieve Elements From An Array :

- An array given i.e. 'int[ ] a={10,20,30};', now we can print elements of an array by two ways, which are as follows :

Way 1 : (using for loop)

```
for(int i=0;i<a.length;i++)
{
    System.out.println(a[i]);
}
```

Way 2 : (using for-each loop)

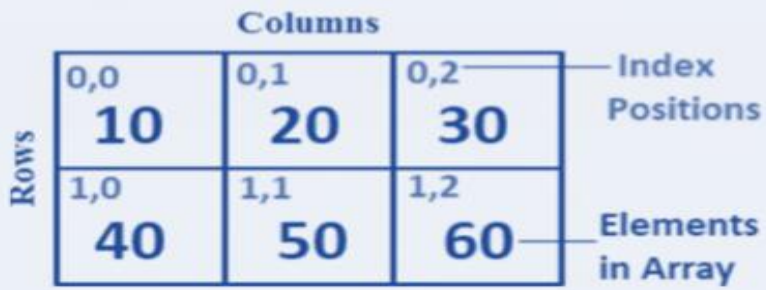
```
for(int i:a)
{
    System.out.println(i);
}
```

Multi-Dimensional (2-D)

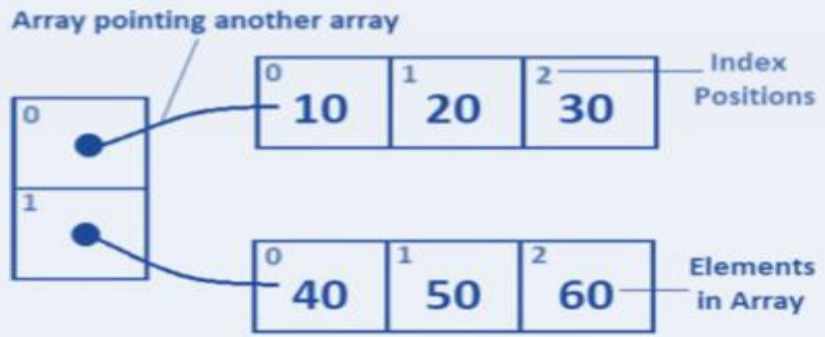
What is Multi-Dimensional Array :

- An array having multiple rows or columns is known as multi-dimensional array.
- These are also known as array of arrays because array is present in another array.
- There are two types of multi-dimensional array :
  1. 2-D Array
  2. 3-D Array

- We can represent 2-D multi-dimensional array as follows :



OR



Multi-Dimensional (2-D)

Declaration Of 2-D Multi-Dimensional Array :

- Diferent Ways of Declaration of Arrays are :
  1. int[ ][ ] a;
  2. int [ ][ ]a;
  3. int[ ][ ]a;
  4. int a[ ][ ];
  5. int[ ] a[ ];

- Most Preferred Declaration is 'int[ ][ ] a;', because here 'a' is two dimensional int array, thus name is clearly separated with type.

- We cannot provide size at the time of array declaration i.e. 'int[2][3] a;' or 'int a[2][3];', this type of any statement is incorrect.

- Note that, there is difference between below statements :

```
int[ ][ ] a,b; // here 'a' and 'b' both are 2-D Arrays.
int[ ] a[ ], b; // here 'a' is 2-D and 'b' is 1-D Array.
int[ ] a[ ], b[ ]; // 'a' and 'b' both are 2-D Array.
int[ ] [ ]a, [ ] b; // compile time error.
int[ ] [ ]a, b[ ]; // 'a' is 2-D and 'b' is 3-D Array.
```