Enumeration

CDAC PATNA

- In Java, enumeration defines a class type. An Enumeration can have constructors, methods and instance variables.
- Enumeration is created by using a keyword called "**enum**".
- Each enumeration constant is *public*, *static* and *final* by default.
- The main objective of enum is to define our own data types.
- Even though enumeration defines a class type and have constructors, we do not instantiate an **enum** using **new keyword**.
- All Enumerations by default inherit **java.lang.Enum** class.

- Syntax-

```
enum  enumerated_type_name
{
        enumerator1, enumerator2;
}

enum  enumerated_type_name
      {
            enumerator1, enumerator2
      }
```

Note: The enum can be defined with in or outside the class.   It should not be inside a method. The semicolon (;) at the end of the enum constants are optional.

- Example –

 we are going to initialize four enumerators i.e. spade, heart, diamond, and club belonging to an enumerated type called **cards**.

- An enumeration can be defined simply by creating a list of **enum** variable. Let us take an example

Enumeration

```
enum cards{
        club,heart,diamond,spade;
}
```

Enumerators (enum constants)

- These values inside the braces are called enum constants i.e club,heart,diamond and spade are called **enumeration constants**. These are public, static and final by default.

Note: The enum constants are usually represented in uppercase.

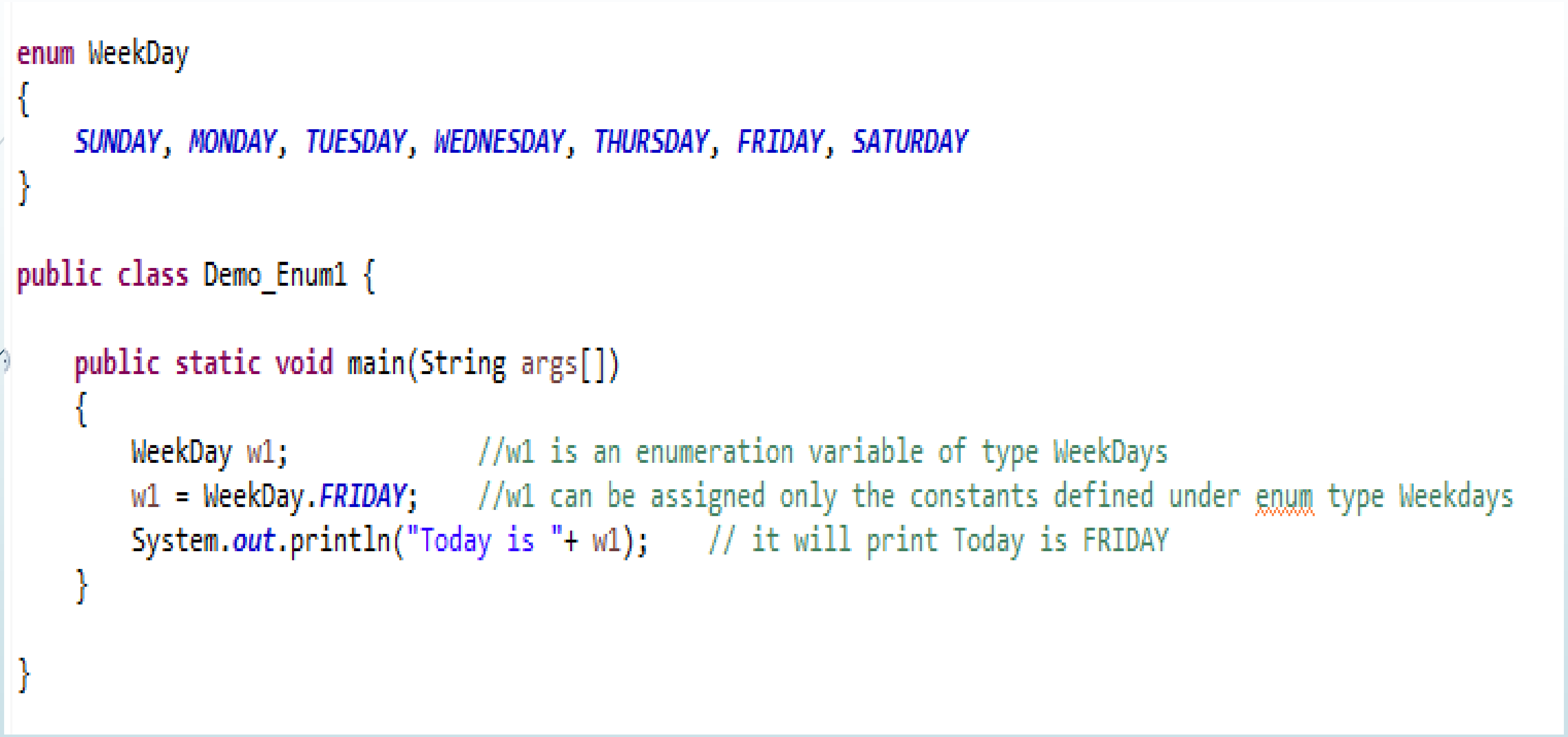
- Object of enumcan be defined directly without **new** keyword

  (cards c1 ; // c1 is an object of enum of type cards)

  (c1 = cards.spade; //c1 can be assigned only the constants defined under enum type **cards**)

or,

cards c1 = cards.spade ;



```
enum WeekDay
{
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

public class Demo_Enum1 {

    public static void main(String args[])
    {
        WeekDay w1;              //w1 is an enumeration variable of type WeekDays
        w1 = WeekDay.FRIDAY;    //w1 can be assigned only the constants defined under enum type Weekdays
        System.out.println("Today is "+ w1);    // it will print Today is FRIDAY
    }

}
```

Enum Class in Java

Every enum in java is child class of java.lang.Enum class i.e every enum always extend the “java.lang.Enum” class of java

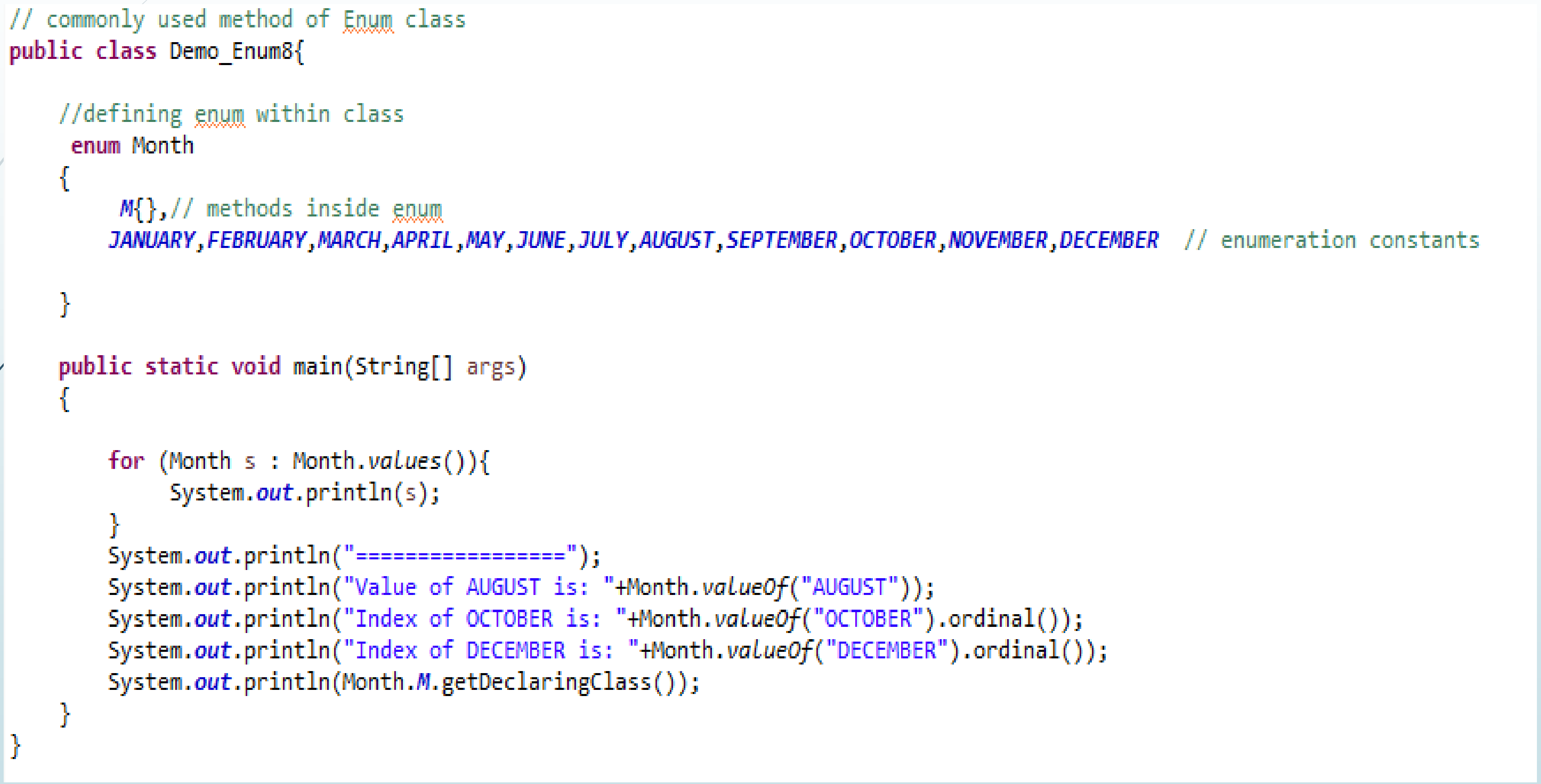
Note- Inheritance concept Is not applicable for our own enum explicitly.

we can say , **extends** keyword is banned for enum

Methods of Enum Class -

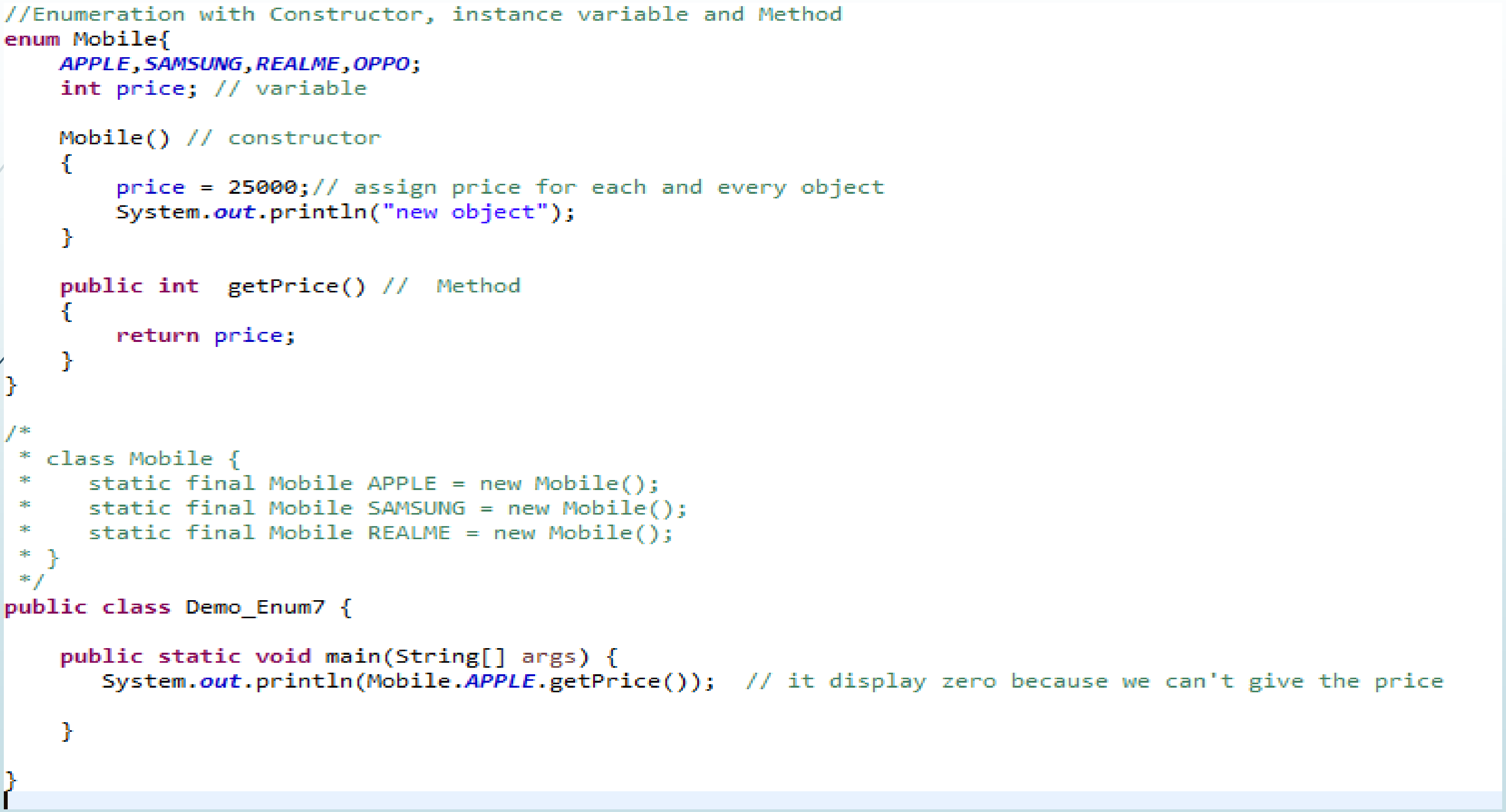
- Values() - used to return all values present inside the enum
- ValueOf() - method returns the enumeration constant of the input String. If the String specified is not matched with the constant then it will throw IllegalArgumentException.
- ordinal() - method returns the order of the enumeration constant based on the index of the constant.
- compareTo() - method compares the enum constants based on their ordinal value
- name() - method returns the defined name of an enum constant in string form.
- toString() - Returns the name of the invoking constant. Not necessarily matches the name used in the enumeration’s declaration.
- getDeclaringClass() - Returns the enumeration type (also known as enum declaring class) of which the invoking constant is a member.

- Example-

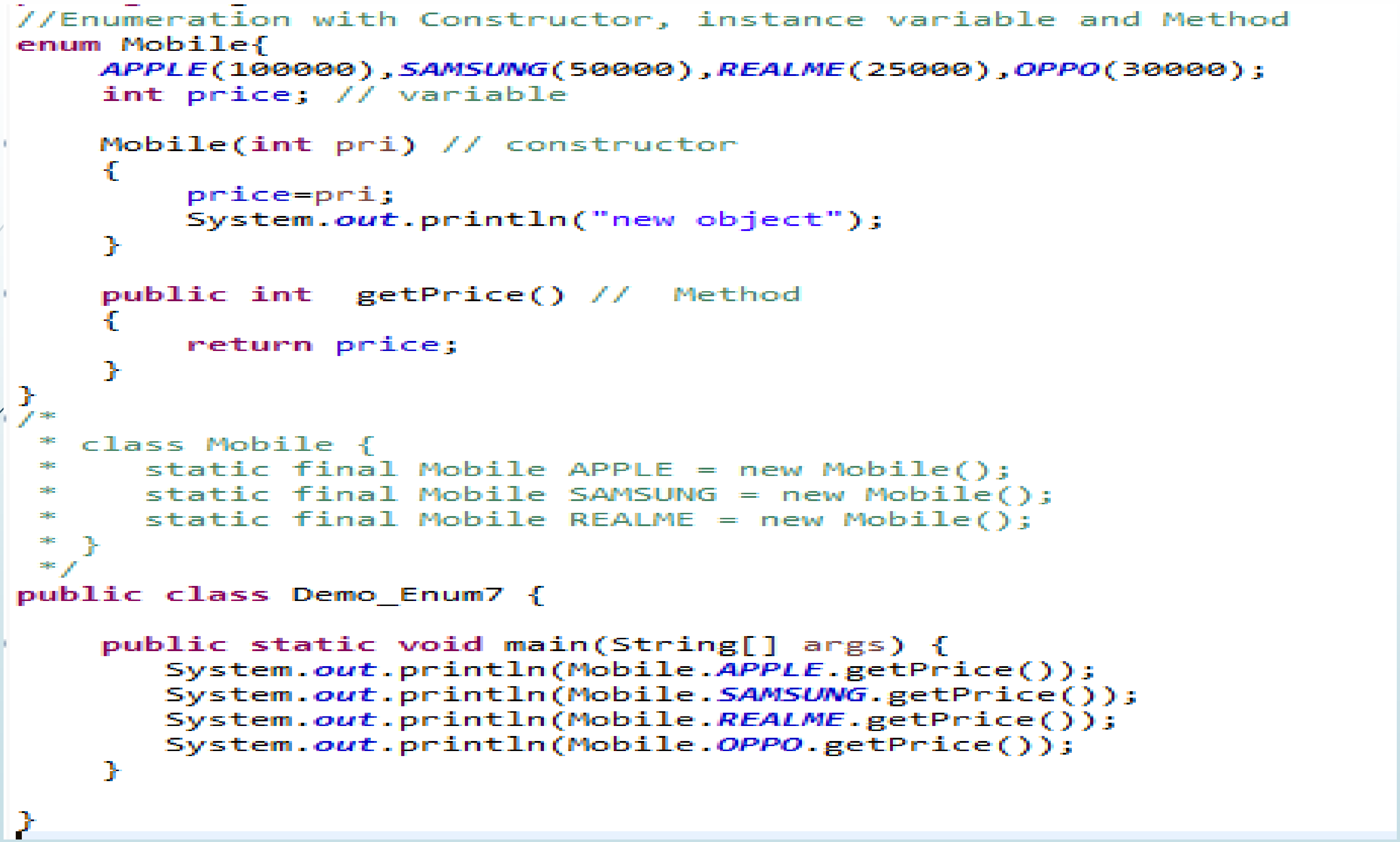


```
// commonly used method of Enum class
public class Demo_Enum8{

    //defining enum within class
     enum Month
    {
         M{},// methods inside enum
        JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER  // enumeration constants

    }

    public static void main(String[] args)
    {

        for (Month s : Month.values()){
             System.out.println(s);
        }
        System.out.println("=================");
        System.out.println("Value of AUGUST is: "+Month.valueOf("AUGUST"));
        System.out.println("Index of OCTOBER is: "+Month.valueOf("OCTOBER").ordinal());
        System.out.println("Index of DECEMBER is: "+Month.valueOf("DECEMBER").ordinal());
        System.out.println(Month.M.getDeclaringClass());
    }
}
```

➨Enumeration with Constructor, instance variable and Method



```
//Enumeration with Constructor, instance variable and Method
enum Mobile{
    APPLE,SAMSUNG,REALME,OPPO;
    int price; // variable

    Mobile() // constructor
    {
        price = 25000;// assign price for each and every object
        System.out.println("new object");
    }

    public int  getPrice() //  Method
    {
        return price;
    }
}

/*
 * class Mobile {
 *    static final Mobile APPLE = new Mobile();
 *    static final Mobile SAMSUNG = new Mobile();
 *    static final Mobile REALME = new Mobile();
 * }
 */
public class Demo_Enum7 {

    public static void main(String[] args) {
       System.out.println(Mobile.APPLE.getPrice());  // it display zero because we can't give the price

    }

}
```



```
//Enumeration with Constructor, instance variable and Method
enum Mobile{
	APPLE(100000),SAMSUNG(50000),REALME(25000),OPPO(30000);
	int price; // variable

	Mobile(int pri) // constructor
	{
		price=pri;
		System.out.println("new object");
	}

	public int  getPrice() //  Method
	{
		return price;
	}
}
/*
 * class Mobile {
 *    static final Mobile APPLE = new Mobile();
 *    static final Mobile SAMSUNG = new Mobile();
 *    static final Mobile REALME = new Mobile();
 * }
 */
public class Demo_Enum7 {

	public static void main(String[] args) {
	   System.out.println(Mobile.APPLE.getPrice());
	   System.out.println(Mobile.SAMSUNG.getPrice());
	   System.out.println(Mobile.REALME.getPrice());
	   System.out.println(Mobile.OPPO.getPrice());
	}

}
```

Inner Class

Inner Class

- An inner class is defined as a class that is declared inside another class.
- The class that holds the inner class is called the **outer class**.
- Inner classes can be either static or non-static.
- Syntax :- The class **OuterDemo** is the outer class and the class **InnerDemo** is the nested class (inner class).

```
class OuterDemo{
        // code of outer class
          class InnerDemo{
               // code of inner class
        }
}
```

Features of Inner class –

- An inner class cannot have the same name as the outer class.
- But, it is possible to use the same names for members(data/ variables or method) of both outer and inner classes.
- Without existing an outer class object or instance, there will be no chance of an existing inner class object.
- The scope of inner class is bounded by the scope of its outer class.
- An inner class can directly access all the variables and methods of the outer class including private.

Syntax to create object of Inner class in Java –

Syntax:

```
        OuterClass.InnerClass  innerObject = outerObject.new   InnerClass();
```

For Example:- OuterClass(A) and InnerClass(B)

```
      // first create an object of outer class (A)
        A outerObj = new A();
      // second create an object of inner class (B)
        A.B  innerObj = outerObj.new  B();
```

- There are two types of nested classes in java :-
- Static
- Non-Static

i. **Static Inner Class :-** A static inner class is one that is declared with the static keyword inside a class .

- It can access static data members of the outer class, including private.
- It cannot access non-static data members and methods.
- It can be accessed by outer class name.
- Syntax -

OuterClassName.InnerClassName object_of_innerclass = new OuterclassName. InnerClassName();

- Example-

ii. Non-Static Inner Class :-

a. Method Local Inner Class :- A non-static class i. e declared within a method of the outer class is called method local inner class.

- If you want to invoke the methods of the local inner class, you must create object of local inner class inside the method (i. e method of outer class).

b. Normal or Regular Inner Class :- A non-static class that is created inside a class but outside a method is called member inner class. It is also known as a regular inner class.

c. Anonymous inner class :- An inner class declared without a class name is known as an anonymous inner class and for which only a single object is created.

- Both the declaration and instantiation of an anonymous inner class occur simultaneously.

- Syntax of an anonymous inner class :-

```
 AnonymousInner   obj_name  = new  AnonymousInner()
{
      void method_name ()
       {
              …………………
       }
};
```

How to take input from user

- Java provides three classes: BufferedReader, Scanner and Console - to take input from user.

1. **Scanner-** Java **Scanner class** allows the user to take input from the user. It belongs to **java.util** package. It is the easiest way to take input from user.

- To use the Scanner we need to import the Scanner Class i.e belongs to java.util package.

  Syntax- import java.util.Sacnner;

- Then, we need to create an object of the Scanner Class with the new.We can use the object to take input from the user.

  Syntax- Scanner object_name = **new** Scanner(System.in);

  Example- Scanner sc = new Scanner(System.in);

Methods of Scanner Class

Method	Description
nextInt()	int value from the user
nextByte()	byte value from the user
nextShort()	short value from the user
nextLong()	long value from the user
nextFloat()	float value from the user
nextDouble()	double value from the user
nextBoolean()	boolean value from the user
next()	A character or word value from user
nextLine()	String value from the user

2. BufferedReader -Java **BufferedReader class** allows the user to read the stream of characters (string) . It belongs to **java.io** package.(If we input data through this class but we have to handle the exception or throws the exception)

- **Syntax –**

```
    BufferedReader obj_name = new BufferedReader(new
InputStreamReader(System.in));
```

- To use the **BufferedReader** we need to import the BufferedReader Class i.e belongs to java.io package.
- Here we need to import another class i.e **InputStreamReader** i.e also belongs to java.io package.

- **Methods of BufferedReader –**

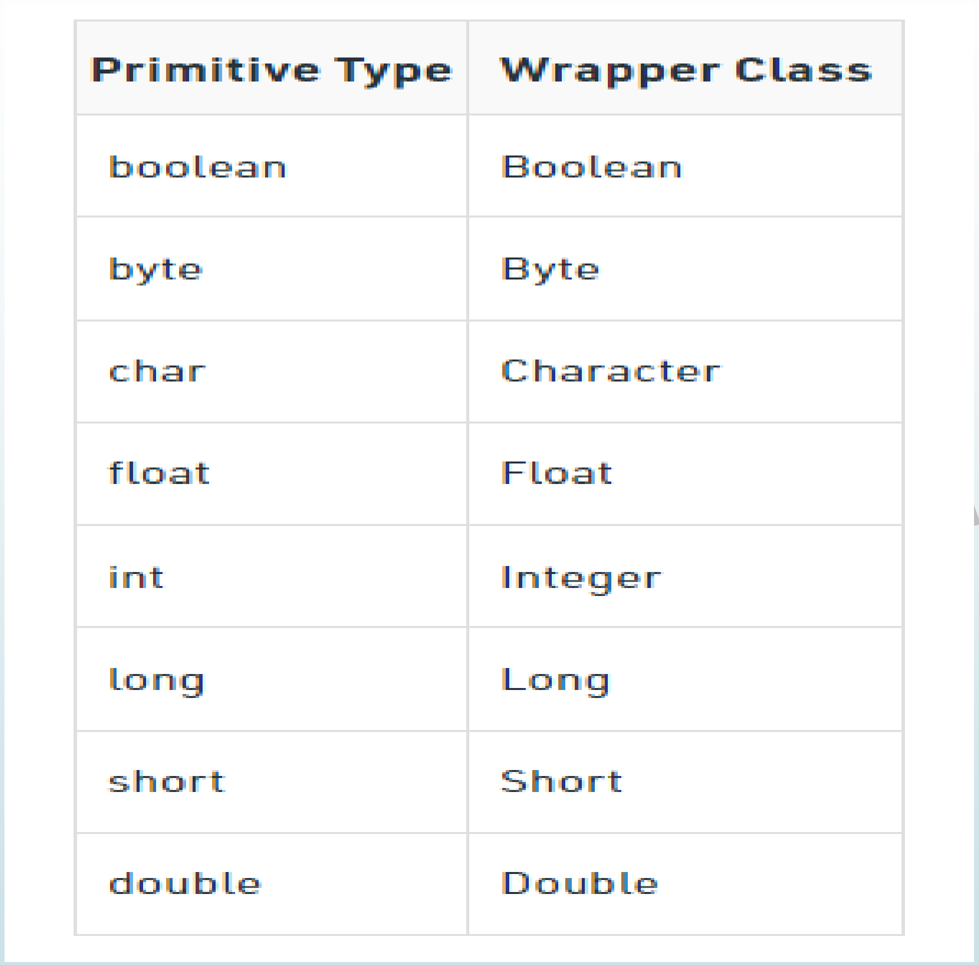
This class provide method **readLine()** which reads the data from user (character/string)

3. Command Line Argument –

- We can provide inputs to our program through the Command Prompt just the moment after we begin to execute our program.
- The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. The command line is given to args[]. **These programs have to be run on cmd**.
- These inputs are to be added just after the filename in the run command. They are taken as strings and passed to the main function as its parameters.

Autoboxing & Unboxing

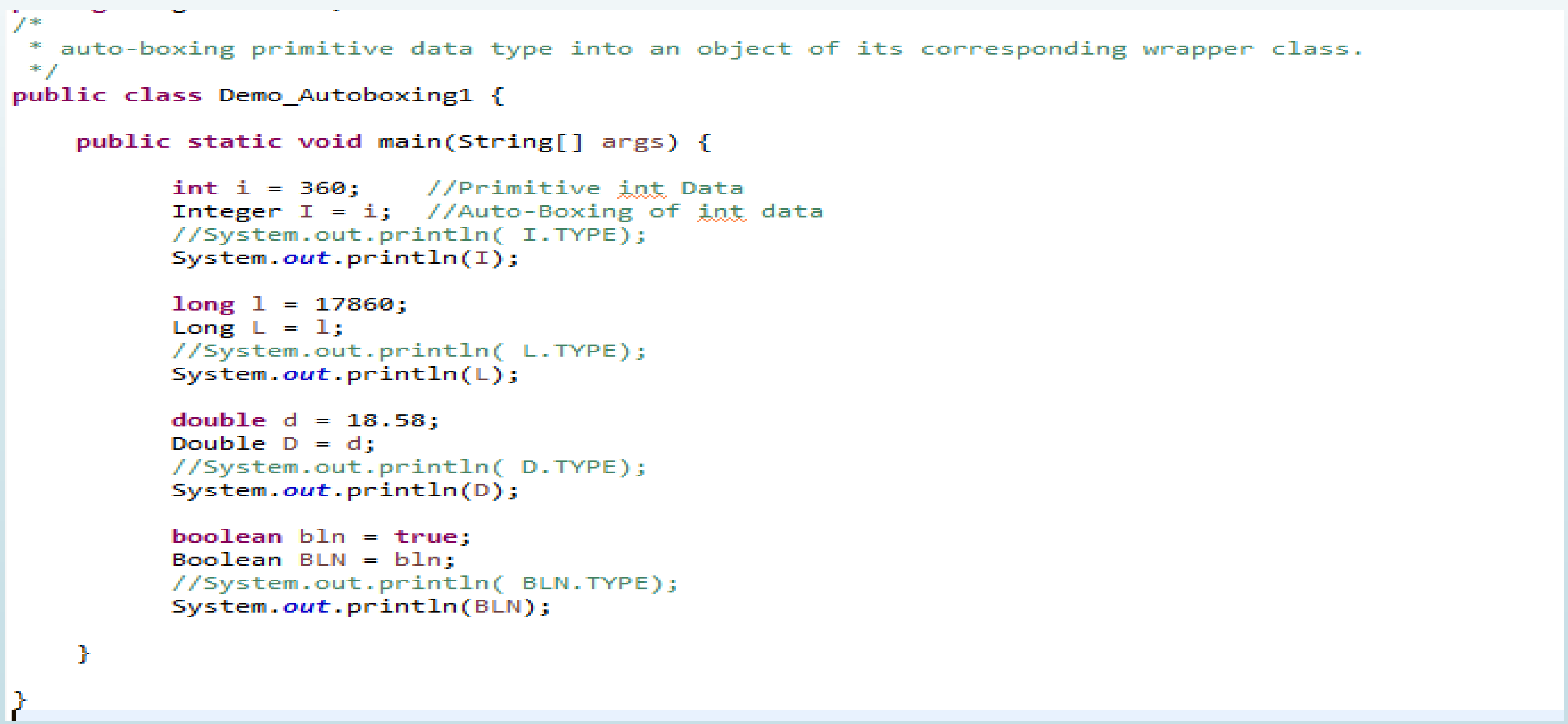
CDAC PATNA



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

Autoboxing -

- Autoboxing is the automatic conversion.
- Autoboxing in Java is a process of converting a ***primitive data type*** into an object of its corresponding wrapper class. For example, converting int to Integer class, long to Long class or double to Double class, etc.



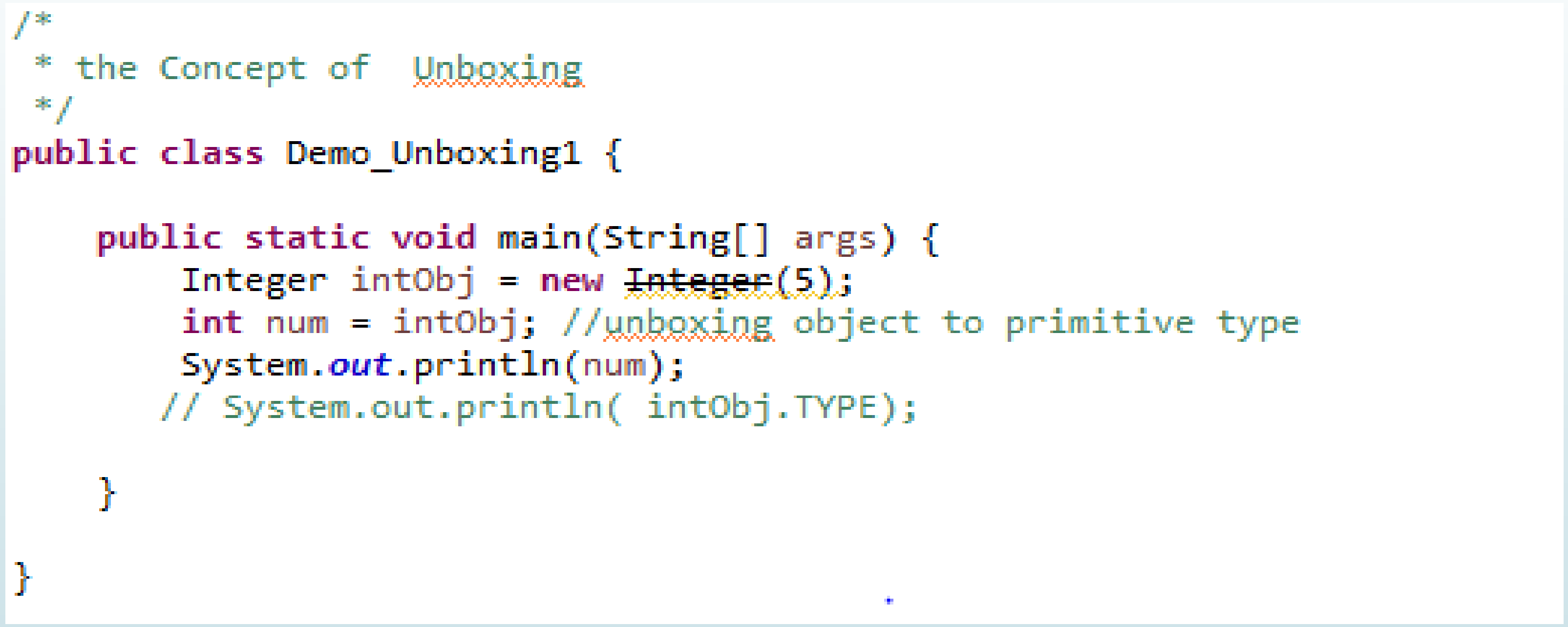
```
/*
 * auto-boxing primitive data type into an object of its corresponding wrapper class.
 */
public class Demo_Autoboxing1 {

    public static void main(String[] args) {

          int i = 360;     //Primitive int Data
          Integer I = i;  //Auto-Boxing of int data
          //System.out.println( I.TYPE);
          System.out.println(I);

          long l = 17860;
          Long L = l;
          //System.out.println( L.TYPE);
          System.out.println(L);

          double d = 18.58;
          Double D = d;
          //System.out.println( D.TYPE);
          System.out.println(D);

          boolean bln = true;
          Boolean BLN = bln;
          //System.out.println( BLN.TYPE);
          System.out.println(BLN);

    }

}
```

Unboxing (Wrapper Objects to Primitive Types)

- Unboxing refers to converting an object of a wrapper type to its corresponding primitive value.
- Unboxing in Java is an automatic conversion of an object of a wrapper class to the value of its respective primitive data type by the compiler.
- It is the opposite technique of Autoboxing. For example, conversion of Integer type to int type or Byte to byte type etc.

Benefits of Autoboxing / Unboxing

- We don't have to perform Explicit **typecasting**.



```
/*
 * the Concept of  Unboxing
 */
public class Demo_Unboxing1 {

    public static void main(String[] args) {
        Integer intObj = new ~~Integer~~(5);
        int num = intObj; //unboxing object to primitive type
        System.out.println(num);
       // System.out.println( intObj.TYPE);

    }

}
```

THANK YOU

CDAC PATNA