2.Object and Classes

1.Introduction

A class represents a type for eg.student,book, etc. We can not store data in a class or perform operation on it,we must create object.An object belongs to class.

2.Defining classes

Syntax:

access_modifier ***class*** *<class_name>*
{
data member;
method;
constructor;
nested class;
interface;
}

Defining methods

Syntax:

```
<access_modifier><return_type><method_name>(
list_of_parameters)
{
//Local declaration
//body
}
```

Types of classes

1.Concrete class

Any normal class which does not have any abstract method.It is a complete class which defines data members,constructors and methods.It can also be sub-class.

2.Abstract class

It is an incomplete class which cannot be instantiated.It defines data members,defines constructors,defines some methods,declares other methods.

3.Final class

A class declared with the final keyword is a final class and it cannot be extended by another class.

4.Static class
Static classes are nested classes means a class declared within another class as a static member is called a static class.

5.Inner Class
A class declared within another class or method is called an inner class.

6.Interface or pure abstract class
It is a pure abstract class. It simply declares the methods.

3.Creating Objects

An object can be created in two ways:
1.Declare a reference of the class.This reference does not create an object.

Syntax: Classname referencename;

Eg	Circle c1;

2.create an object using **new** operator and assign object to the reference.The new operator dynamically allocates memory for the object.

Syntax: referencename = new Classname();

Eg   c1=new Circle();

Or
Classname referencename =new Classname();

Example of class and object

```
public class Main
{
  int x = 5;
  public static void main(String[] args)
{
    Main m1= new Main();
    System.out.println(m1.x);
  }
}
```

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main m1 = new Main();  // Object 1
    Main m2 = new Main();  // Object 2
    System.out.println(m1.x);
    System.out.println(m2.x);
  }
}
```

Main.java

```
public class Main
{
  int x = 5;
}
```

Second.java

```
class Second
 {
  public static void main(String[] args)
{
    Main m1 = new Main();
    System.out.println(m1.x);
  }
}
```

4.Array of Objects

In many cases we need to create several objects of a class.So we can create an array of objects.

Steps to create an array of objects:

1.create an array of reference using new.

Syntax:

```
Classname[] arrayname=new Classname[size];
```

Eg `Circle[] c=new Circle[5];`

2.create an object for each reference using new.

Syntax :

```
 for(i=0;i<size;i++)
arrayname[i]=new Classname();
```

eg `c[i]=new Circle();`

```
public class Main
{
    public static void main(String args[])
    {

        //step1 : first create array of 10 elements that
holds object addresses.
        Emp[] employees = new Emp[10];
        //step2 : now create objects in a loop.
        for(int i=0; i<employees.length; i++){
            employees[i] = new Emp(i+1);//this will call
constructor.
        }
    }
}
class Emp{
    int eno;
    public Emp(int no){
        eno = no;
        System.out.println("emp constructor called..eno
is.."+eno);
    }
}
```

5.Constructor

A constructor is a special method of the class which has the same name as the class. It is used for initializing the data members of an object.

Rules:

1. Must have same name as the class.
2. Does not have any return type not even void.
3. Zero or more arguments.
4. Class can have more than one constructor.
5. Cannot be declare static,final.
6. Can be declare public ,private or protected.
7. It is called automatically when as object of class is created.
8. Constructors are not inherited.

Types of Constructors

1. Default constructor
   A constructor is called "Default Constructor" when it does not have any parameter.

Syntax:

```
<class_name>(){}
```

Example:

```
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();
}
}
```

2. Parameterized constructor

   A constructor which has a specific number of parameters is called a parameterized constructor.

Syntax :

```
<classname>(para1,para2){}
```

Example:

```
class Student
{
    int id;
    String name;
    //creating a parameterized constructor
    Student(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
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    //calling method to display the values of
object
    s1.display();
    s2.display();
   }
}
```

this keyword

The this keyword in Java is a reference variable that refers to the current object. It's used in constructors and instance methods to access the current object's members.

Use cases

Access members: Use this to access the current object's instance variables, methods, and constructors

Reduce confusion: Use this to differentiate between class attributes and parameters with the same name

Call another constructor: Use this to call another constructor in the same class

Pass as an argument: Use this as an argument in constructor calls.

6. Static Fields, Methods and Blocks

Static Fields

In Java, static fields, also known as class variables, are variables that are shared across all instances of a class. They are declared using the static keyword.

```
public class MyClass
 {
   public static int myStaticField = 10;
}
```

Static fields are initialized once and are stored in a fixed location in memory.
They can be accessed directly via the class name.
They can be used to maintain global state or constants.

Static Methods

A static method in Java is a method that belongs to a class, not an object. It can be called directly using the class name, without creating an object of the class.

```
Access_modifier static void methodName()
{
    // Method body.
}
```

The name of the class can be used to invoke or access static methods.

```
className.methodName();
```

Static Blocks

In a Java class, a static block is a set of instructions that is run only once when a class is loaded into memory. A static block is also called a static initialization block.

```
public class StaticBlock
{

// print method of the StaticBlock class
public static void print()
{
System.out.println("Inside the print method.");
}
```

```
static
{
System.out.println("Inside the static block.");
}

// main method
public static void main(String[] args)
{

// instantiating the class StaticBlock
StaticBlock sbObj = new StaticBlock();
sbObj.print(); // invoking the print() method
  }
```

7.Predefined Classes

7.1 The object class

The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

Object class in Java is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class.

Object Class Methods

The Object class provides multiple methods which are as follows:

toString() method
hashCode() method
equals(Object obj) method
finalize() method
getClass() method
clone() method
wait(), notify() notifyAll()

1. toString() Method
The toString() provides a String representation of an object and is used to convert an object to a String. The default toString() method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@’, and the unsigned hexadecimal representation of the hash code of the object.
The toString() method is overridden to return a custom string representation of the object.

```
public class Main
{
  public static void main(String[] args)
```

```
    {
    Main m=new Main();
    System.out.println(m.toString());
      String myStr = "Hello, World!";
      System.out.println(myStr.toString());

    }
  }
```

2. hashCode() Method

For every object, JVM generates a unique number which is a hashcode.

```
public class Main {

    public static void main(String[] args)
    {
          Main m=new Main();
            System.out.println(m.hashCode());
              String myStr = "Hello";
              System.out.println(myStr.hashCode());

    }

}
```

3. equals() Method

The equals() method compares two strings, and returns true if the strings are equal, and false if not.

```
public class Main {

    public static void main(String[] args)
    {
            String myStr1 = "Hello";
              String myStr2 = "Hello";
              String myStr3 = "Another String";
              System.out.println(myStr1.equals(myStr2));
              System.out.println(myStr1.equals(myStr3));

    }

}
```

4. getClass() Method

Object getClass() method returns the runtime class of an object.

It returns an instance of the Class class, which provides methods to inspect the properties of the class, such as its name, superclass, interfaces, constructors, methods, and fields.

```
public class Main {

        public static void main(String[] args)
        {
                Main m=new Main();
           Class c=m.getClass();
         System.out.println(" " +c);
         System.out.println(" " + c.getName());
         System.out.println(" " + c.getCanonicalName());
         System.out.println(" " + c.getModifiers());
         System.out.println(" " + c.getPackageName());
         System.out.println(" " + c.getSimpleName());
         System.out.println(" " + c.getTypeName());

        }

}
```

7.2 String class

String is a sequence of characters. In Java, objects of the **String class** are immutable which means they cannot be changed once created.

Creating a String

There are two ways to create string in Java:

1. Using String literal

```
String s = “hello”;
```

2. Using new keyword

String s = new String (“hello”);

String Constructors in Java

String();
Eg String s=new String();

String(String obj);
Eg String s1=new String();
String s2=new String(s1);

String(char[] arr)
Eg char[] arr={‘a’,’b’,’c’,’d’,’e’};
String s=new String(arr);

String(char[] arr,int startindex,int numchars);
Eg char[] arr={‘a’,’b’,’c’,’d’,’e’};
String s=new String(arr,1,3);

String(byte[] arr)
Eg byte[] arr={65,66,67,68,69,70};
String s=new String(arr);

String(byte[] arr,int startindex,int numchars);
Eg byte[] arr={65,66,67,68,69,70};
String s=new String(arr,1,3);

String Operations

Java String length()
Returns the length of the string

Java String replace()
Replace all matching characters/text in the string

Java String replaceAll()
Replace all substrings matching the regex pattern

Java String substring()
Returns a substring from the given string

Java String equals()
Compares two strings

Java String indexOf()
Returns the index of the character/substring

Java String trim()
Removes any leading and trailing whitespace

Java String charAt()
Returns the character at the given index

Java String toLowerCase()
Converts characters in the string to lower case

Java String concat()
Concatenates two strings and returns it

Java String startsWith()
Checks if the string begins with the given string

Java String endsWith()
Checks if the string ends with the given string

Java String isEmpty()
Checks whether a string is empty or not

Java String format()
Returns a formatted string

7.3 StringBuffer class

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

StringBuffer Constructors

StringBuffer()
It reserves room for 16 characters without reallocation
StringBuffer s = new StringBuffer();

StringBuffer(int size)
It accepts an integer argument that explicitly sets the size of the buffer.
StringBuffer s = new StringBuffer(20);

StringBuffer(String str)
It accepts a string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.
StringBuffer s = new StringBuffer(“hello”);

StringBuffer Methods

append() Used to add text at the end of the existing text.
length() The length of a StringBuffer can be found by the length( ) method.
capacity() the total allocated capacity can be found by the capacity( ) method.
charAt() This method returns the char value in this sequence at the specified index.
delete() Deletes a sequence of characters from the invoking object.
deleteCharAt() Deletes the character at the index specified by the loc.
ensureCapacity() Ensures capacity is at least equal to the given minimum.
insert() Inserts text at the specified index position.
length() Returns the length of the string.
reverse() Reverse the characters within a StringBuffer object.
replace() Replace one set of characters with another set inside a StringBuffer object.

7.4 Wrapper classes

A Wrapper class in Java is one whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Primitive Data Types and their Corresponding Wrapper Class

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

Creating Wrapper Objects
To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

Example
```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Convert Primitive Type to Wrapper Objects
We can also use the valueOf() method to convert primitive types into corresponding objects.

```
class Main {
  public static void main(String[] args) {
    // create primitive types
    int a = 5;
    double b = 5.65;

    //converts into wrapper objects
    Integer aObj = Integer.valueOf(a);
    Double bObj = Double.valueOf(b);

    if(aObj instanceof Integer) {
      System.out.println(""+aObj);
    }

    if(bObj instanceof Double) {
      System.out.println(""+bObj);
    }
  }
}
```

instanceof operator to check whether the generated objects are of Integer or Double type or not.

the Java compiler can directly convert the primitive types into corresponding objects. For example,

```
int a = 5;
// converts into object
Integer aObj = a;
```

```
double b = 5.6;
// converts into object
Double bObj = b;
```

Wrapper Objects into Primitive Types

To convert objects into the primitive types, we can use the corresponding value methods (intValue(), doubleValue(), etc) present in each wrapper class.

```
class Main {
  public static void main(String[] args) {

    // creates objects of wrapper class
    Integer aObj = Integer.valueOf(23);
    Double bObj = Double.valueOf(5.55);

    // converts into primitive types
    int a = aObj.intValue();
    double b = bObj.doubleValue();

    System.out.println("The value of a: " + a);
    System.out.println("The value of b: " + b);
  }
}
```

the Java compiler can automatically convert objects into corresponding primitive types. For example,

```
Integer aObj = Integer.valueOf(2);
// converts into int type
int a = aObj;

Double bObj = Double.valueOf(5.55);
// converts into double type
double b = bObj;
```

8.Packages

Packages in Java are a mechanism that encapsulates a group of classes, sub-packages, and interfaces.

1. Built-in Packages

These packages consist of a large number of classes which are a part of Java API.Some of the commonly used built-in packages are:

java.lang: Contains language support classes(e.g classes which defines primitive data types, math operations). This package is automatically imported.

java.io: Contains classes for supporting input / output operations.

java.util: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.

java.applet: Contains classes for creating Applets.

java.awt: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

java.net: Contain classes for supporting networking operations.

2. User-defined Packages
These are the packages that are defined by the user.

Create the Package:
Syntax: package packagename;

Accessing the Package:
Syntax: packagename.classname;

There are three ways to access the package from outside the package.

import package.*;
import package.classname;
fully qualified name.