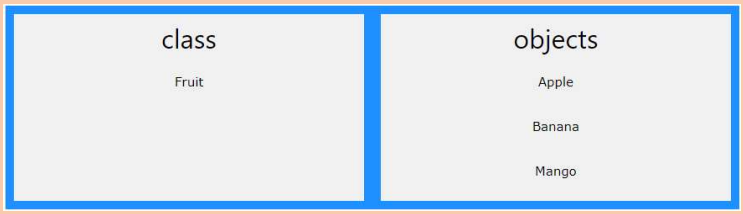
CHAPTER 2

OBJECT ORIENTED PROGRAMMING IN JAVA

Class Fundamentals

❖Classes and Objects

- Classes and objects are the two main aspects of object-oriented programming.



- So, a class is a template for objects, and an object is an instance of a class.
- A Class is a "blueprint" for creating objects.

❖Create a Class

- To create a class, use the keyword class

❖Create an Object

- To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new

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

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

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

❖Multiple Objects

- You can create multiple objects of one class

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

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

❖ Initialize the object through a reference variable

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

Output

101 sonoo

❖Using Multiple Classes

- We can also create an object of a class and access it in another class.
- This is often used for better organization of classes
- One class has all the attributes and methods, while the other class holds the main() method (code to be executed).
- Remember that the name of the java file should match the class name.
- In the following example, we have created two files in the same directory/folder:

  MyClass.java

  OtherClass.java

MyClass.java

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

OtherClass.java

```
class OtherClass {
  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```


JAVA CLASS ATTRIBUTES

- Class attributes are variables within a class

Example

Create a class called "MyClass" with two attributes x and y

```
public class MyClass {
  int x = 5;
  int y = 3;
}
```

- Another term for class attributes is fields / Data members

❖Accessing Attributes

- We can access attributes by creating an object of the class, and by using the dot syntax (.)

Example

Create an object called "myObj" and print the value of x

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

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Output

5

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

❖Modify Attributes

- We can also modify attribute values

Example

Set the value of x to 40

```
public class MyClass {
  int x;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Output

40

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

❖Override existing values

Example

Change the value of x to 25

```
public class MyClass {
  int x = 10;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Output

25

- If you don't want the ability to override existing values, declare the attribute as **final**

```
public class MyClass {
  final int x = 10;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

- The final keyword is useful when we want a variable to always store the same value, like PI (3.14159...)
- The final keyword is called a "modifier".

❖Multiple Objects

➢If we create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other

Example - Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged

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

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Multiple Attributes

➢We can specify as many attributes as you want

```
public class Person {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Person myObj = new Person();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

JAVA CLASS METHODS

- Methods are declared within a class, and that they are used to perform certain actions

Example - Create a method named myMethod() in MyClass

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

- myMethod() prints a text (the action), when it is called.
- To call a method, write the method's name followed by two parentheses () and a semicolon;

Example

Inside main, call myMethod()

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "Hello World!"
```

❖Static vs. Non-Static Methods

- Java programs have either static or public attributes and methods.
- Static method can be accessed without creating an object of the class
- Public methods can only be accessed by objects

Example

The differences between static and public methods


```
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Access Methods With an Object

```
public class Car {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Car myCar = new Car();      // Create a myCar object
    myCar.fullThrottle();       // Call the fullThrottle() method
    myCar.speed(200);           // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```


Remember that..

- The dot (.) is used to access the object's attributes and methods.
- To call a method in Java, write the method name followed by a set of parentheses (), followed by a semicolon (;)

Using Multiple Classes

- It is a good practice to create an object of a class and access it in another class.
- Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

  Car.java

  OtherClass.java

Car.java

```
public class Car {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

OtherClass.java

```
class OtherClass {
  public static void main(String[] args) {
    Car myCar = new Car();      // Create a myCar object
    myCar.fullThrottle();       // Call the fullThrottle() method
    myCar.speed(200);           // Call the speed() method
  }
}
```


CONSTRUCTORS

- A constructor in Java is a **special method** that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes

Types of Java constructors

Default constructor (no-argument constructor)

Parameterized constructor

Syntax of default constructor:

<class_name>(){}

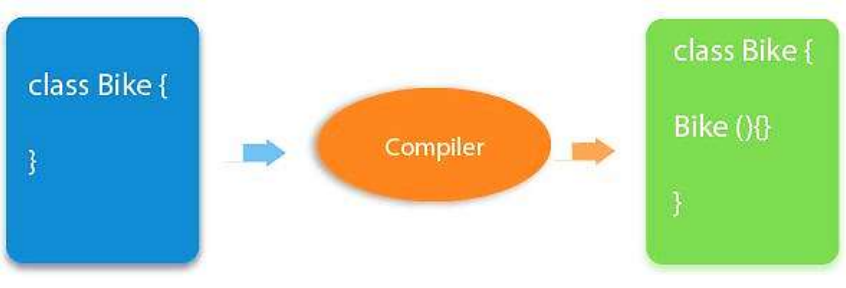
➢In the following example, we are creating the no-argument constructor in the Bike class. It will be invoked at the time of object creation.

```
//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();
}
}
```

Output

Bike is created

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



The purpose of a default constructor

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```java
// Create a MyClass class
public class MyClass {
  int x;  // Create a class attribute

  // Create a class constructor for the MyClass class
  public MyClass() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(); // Create an object of class MyClass (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}

// Outputs 5
```


- The constructor name must match the class name, and it cannot have a return type (like void).
- The constructor is called when the object is created.
- All classes have constructors by default
- If you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

Constructor Parameters (Parameterized constructor)

- Constructors can also take parameters, which is used to initialize attributes

```
public class MyClass {
  int x;

  public MyClass(int y) {
    x = y;
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(5);
    System.out.println(myObj.x);
  }
}

// Outputs 5
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Constructor Parameters - We can have as many parameters as you want

```
public class Car {
  int modelYear;
  String modelName;

  public Car(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Car myCar = new Car(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}

// Outputs 1969 Mustang
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Method Overloading

- With **method overloading**, multiple methods can have the same name with different parameters
- Method overloading is one of the ways that Java supports polymorphism.

There are two ways to overload the method in java

By changing number of arguments

By changing the data type

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Example

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

❖ Advantage of method overloading

- The main advantage of this is cleanliness of code.
- Method overloading increases the readability of the program.
- Flexibility

Example - Consider the following example, which have two methods that add numbers of different type

```
static int plusMethodInt(int x, int y) {
  return x + y;
}

static double plusMethodDouble(double x, double y) {
  return x + y;
}

public static void main(String[] args) {
  int myNum1 = plusMethodInt(8, 5);
  double myNum2 = plusMethodDouble(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

Instead of defining two methods that should do the same thing, it is better to overload one.

```
static int plusMethod(int x, int y) {
  return x + y;
}

static double plusMethod(double x, double y) {
  return x + y;
}

public static void main(String[] args) {
  int myNum1 = plusMethod(8, 5);
  double myNum2 = plusMethod(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

RECURSION

- Recursion is the technique of making a method call itself.
- This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Syntax

```
returntype methodname(){
//code to be executed
methodname();//calling same method
}
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

```
public class MyClass {
  public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
  }
  public static int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
}
```

Working

```
10 + sum(9)
10 + ( 9 + sum(8) )
10 + ( 9 + ( 8 + sum(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

Example

Use recursion to add all of the numbers up to 10.


```
public class RecursionExample3 {
   static int factorial(int n){
       if (n == 1)
        return 1;
       else
        return(n * factorial(n-1));
   }

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Working

```
factorial(5)
   factorial(4)
      factorial(3)
         factorial(2)
            factorial(1)
               return 1
            return 2*1 = 2
         return 3*2 = 6
      return 4*6 = 24
   return 5*24 = 120
```

Example

Factorial of a number

USING OBJECT AS A PARAMETER / ARGUMENT

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }


 public static void main(String args[]){
   Operation2 op=new Operation2();

   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

```
Output:before change 50
       after change 150
```


THIS KEYWORD

➢There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

Usage of java this keyword

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output

```
0 null 0.0
0 null 0.0
```

Understanding the problem without this keyword


```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output

```
111 ankit 5000
112 sumit 6000
```

Solution of the problem with this keyword

- It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

➢ **this: to invoke current class method**

- You may invoke the method of the current class by using the this keyword.
- If you don't use the this keyword, compiler automatically adds this keyword while invoking the method


```
class A{
void m(){System.out.println("hello m");}
void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}}
```

Output

```
hello n
hello m
```

JAVA INNER CLASS

- In Java, it is also possible to nest classes (a class within a class).
- The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.
- To access the inner class, create an object of the outer class, and then create an object of the inner class


```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class MyMainClass {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

➢Access Outer Class From Inner Class

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class MyMainClass {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```

One advantage of inner classes, is that they can access attributes and methods of the outer class

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

Command-Line Arguments

- Sometimes we want to pass information into a program when we run it. This is accomplished by passing command-line arguments to main( ).
- The main method can receive string arguments from the command line
- To access the command-line arguments inside a Java program is quite easy— they are stored as strings in a String array passed to the args parameter of main( ).
- The first command-line argument is stored at args[0], the second at args[1], and so on.

Prepared By Mr. EBIN PM, AP, IESCE EDULINE

```
// Display all command-line arguments.
class CmdLine {
public static void main(String args[]) {
        for(int i=0; i<args.length; i++)
        System.out.println("args[" + i + "]: " +args[i]);
        }
}
```

```
C:\Users\Hello World\Desktop\JAVA>javac CmdLine.java

C:\Users\Hello World\Desktop\JAVA>java CmdLine This is Commend-Line Argument Example
args[0]: This
args[1]: is
args[2]: Commend-Line
args[3]: Argument
args[4]: Example

C:\Users\Hello World\Desktop\JAVA>
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE