# L26

Friday, January 28, 2022 1:18 AM

## Objects, Encapsulation, Different types of Variables, Void

## Using objects

Q1. An instance of a class in memory during program execution is called an object. It encapsulates state information using fields and behaviour using methods.

Instances of a class are created by using the new keyword followed by the constructor. For example:

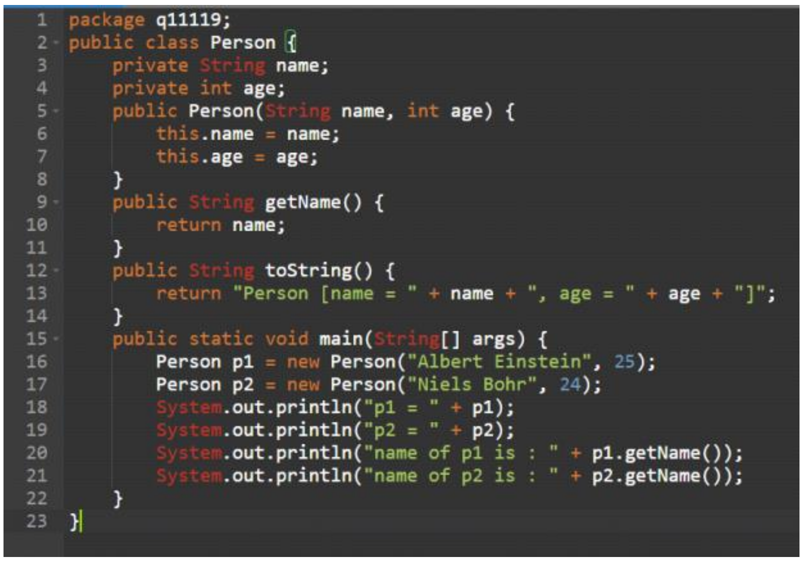
```
String name = new String("Albert Einstien");
Date today = new Date();
Integer luckyNumber = new Integer(7);
```

In the above code name, today and luckyNumber are called ***references*** which store the address of the objects created in memory during execution.

In Java, dot . is used to access a member (field or a method) of an object. For example:

```
name.toLowerCase();
name.substring(6);
```

See and retype the below code, where two instances of Person class are created and stored in references named p1 and p2. Also notice the way we call the method getName() using the dot . on the references.



```java
package q11119;
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public String toString() {
        return "Person [name = " + name + ", age = " + age + "]";
    }
    public static void main(String[] args) {
        Person p1 = new Person("Albert Einstein", 25);
        Person p2 = new Person("Niels Bohr", 24);
        System.out.println("p1 = " + p1);
        System.out.println("p2 = " + p2);
        System.out.println("name of p1 is : " + p1.getName());
        System.out.println("name of p2 is : " + p2.getName());
    }
}
```

Q2. Write a Java program to implement a Class mechanism

Write a Java program with a class name Employee which contains the data members **name** (String), **age** (int), **designation** (String), **salary** (double) and the methods **setData()**, **displayData()**.

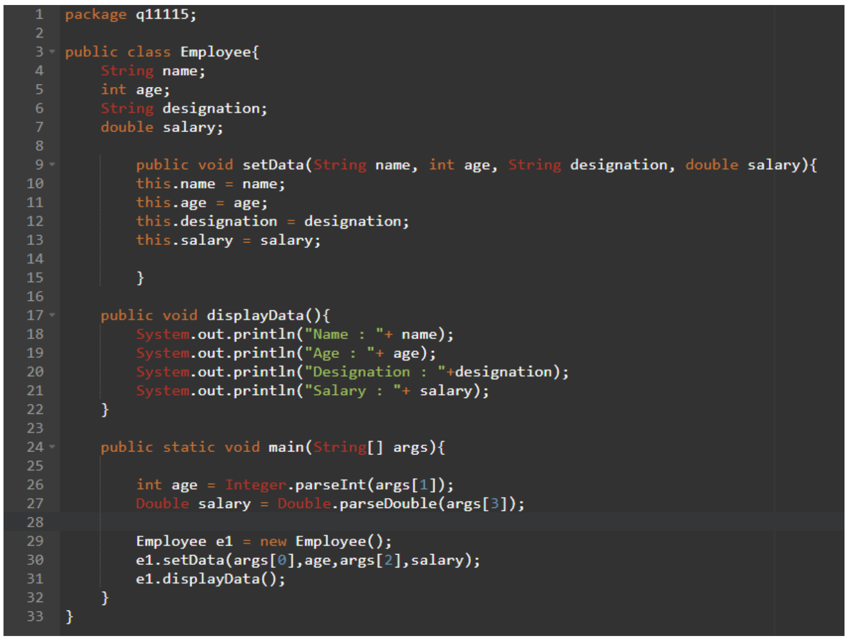
The member function **setData()** is used to initialize the data members and **displayData()** is used to display the given employee data.

Write the **main()** method with in the class which will receive four arguments as **name, age, designation**and **salary**.

Create an object to the class Employee within the main(), call **setData()** with arguments and finally call the method **displayData()** to print the output.

If the input is given as command line arguments to the **main()** as **"Saraswathi", "27", "Teacher", "37250"** then the program should print the output as:

```
Name : Saraswathi
Age : 27
Designation : Teacher
Salary : 37250.0
```



```java
package q11115;

public class Employee{
    String name;
    int age;
    String designation;
    double salary;

        public void setData(String name, int age, String designation, double salary){
        this.name = name;
        this.age = age;
        this.designation = designation;
        this.salary = salary;

        }

    public void displayData(){
        System.out.println("Name : "+ name);
        System.out.println("Age : "+ age);
        System.out.println("Designation : "+designation);
        System.out.println("Salary : "+ salary);
    }

    public static void main(String[] args){

        int age = Integer.parseInt(args[1]);
        Double salary = Double.parseDouble(args[3]);

        Employee e1 = new Employee();
        e1.setData(args[0],age,args[2],salary);
        e1.displayData();
    }
}
```

## Q3. Write a Java program to implement a Constructor

Write a Java program with a class name Staff, contains the data members **id** (int), **name** (String) which are initialized with a **parameterized constructor** and the method **show()**.

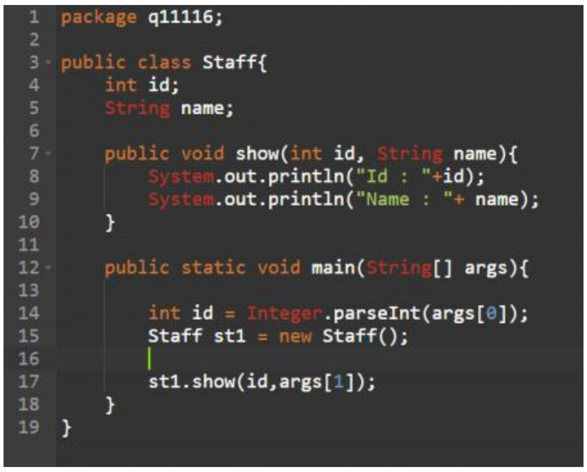
The member function **show()** is used to display the given staff data.

Write the **main()** method with in the class which will receive two arguments as **id** and **name**.

Create an object to the class Staff with arguments **id** and **name** within the **main()**, and finally call the method **show()** to print the output.

If the input is given as command line arguments to the **main()** as **"18", "Gayatri"** then the program should print the output as:

```
Id : 18
Name : Gayatri
```



```java
package q11116;

public class Staff{
    int id;
    String name;

    public void show(int id, String name){
        System.out.println("Id : "+id);
        System.out.println("Name : "+ name);
    }

    public static void main(String[] args){

        int id = Integer.parseInt(args[0]);
        Staff st1 = new Staff();

        st1.show(id,args[1]);
    }
}
```

## Q4. Write a Java program to Access the instance variables by using this keyword

Write a Java program to demonstrate how to access instance variables by using this keyword.

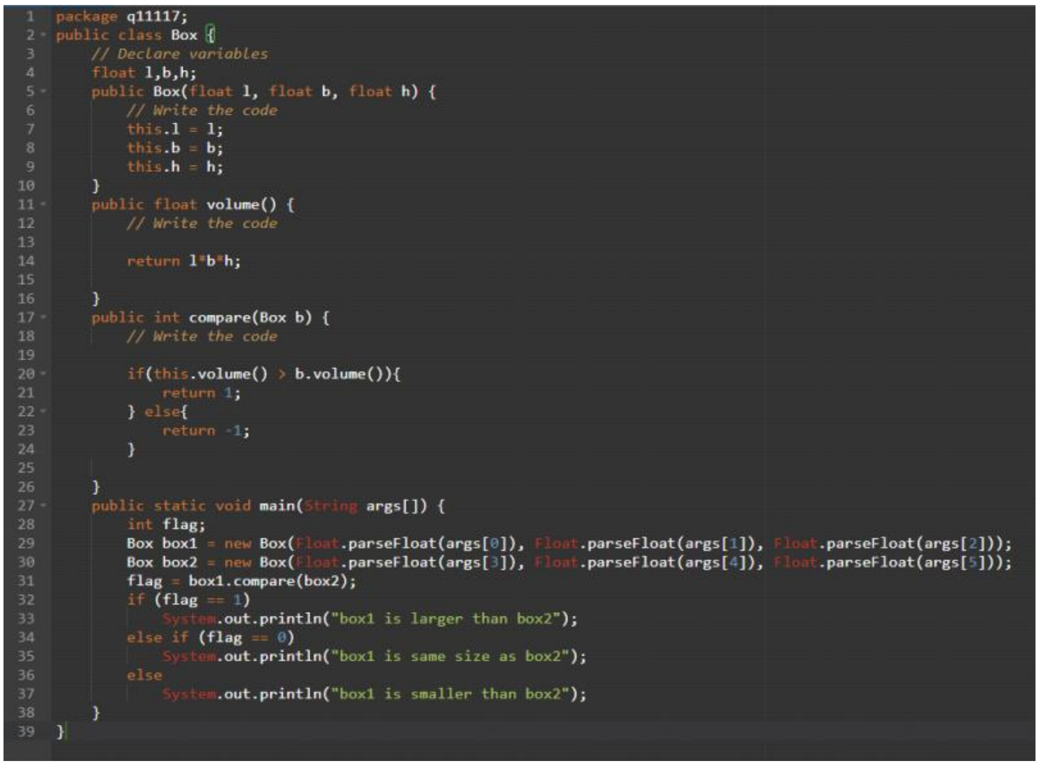
Create a class Box with

- **length**, **breadth** and **height** as data members of type **float**
- parameterized constructor with **three** arguments to initialize the data members with the argument values
- method to find the volume of the box
- method to compare the volumes of two boxes

Write a **main()** function to create objects of the class and to compare the objects.

If the input is given as command line arguments to the **main()** as **["2.5", "3.4", "2.9", "1.6", "2.1", "1.9"]** then the program should print the output as:

```
box2 is smaller than box1
```



```java
package q11117;
public class Box {
    // Declare variables
    float l,b,h;
    public Box(float l, float b, float h) {
        // Write the code
        this.l = l;
        this.b = b;
        this.h = h;
    }
    public float volume() {
        // Write the code

        return l*b*h;

    }
    public int compare(Box b) {
        // Write the code

        if(this.volume() > b.volume()){
            return 1;
        } else{
            return -1;
        }

    }
    public static void main(String args[]) {
        int flag;
        Box box1 = new Box(Float.parseFloat(args[0]), Float.parseFloat(args[1]), Float.parseFloat(args[2]));
        Box box2 = new Box(Float.parseFloat(args[3]), Float.parseFloat(args[4]), Float.parseFloat(args[5]));
        flag = box1.compare(box2);
        if (flag == 1)
            System.out.println("box1 is larger than box2");
        else if (flag == 0)
            System.out.println("box1 is same size as box2");
        else
            System.out.println("box1 is smaller than box2");
    }
}
```

# Encapsulation

## Q1. What is a package and how to use it?

In Java related classes and interfaces can be stored together in folders known as packages.

For example in Java, all the common classes like String, System etc which we have used are in a package called java.lang.

There are many such packages in Java, for example:

1. java.util - this package contains utility and data structure related classes like Date, ArrayList, HashSet, etc., which we will learn later.
2. java.io - this package contains classes which help in reading input and writing output to files and streams.
3. java.net - this package contains networking related classes.

If we want to use a Date class which is in java.util package in our class, we need to inform Java compiler that we are interested in using the Date class of java.util package.

This can be done in the below three ways, either by using an import statement or fully qualified class name:

1. By importing all class of java.util package - import java.util.*; - this statement should occur above the class declaration.
2. By importing only the required class of java.util package - import java.util.Date; - this statement should occur above the class declaration.
3. By directly using the fully qualified name of the class in the declaration statement - java.util.Date date = new java.util.Date(); - we need not have an import when we do this.

**Note:** java.lang package which contains all commonly used Java classes is by default imported. Which means for using a String or an Integer or the System class and the like, one need not import the java.lang package.

Java allows for custom packages to organize the classes. We need to use a package statement like below at the top of a source file to indicate that our class belongs to a certain package. For example:

```
package com.mycompany.games;
public class Cricket {
        ...
}
```

**Note:** While learning, in all our examples **we will not use package declarations**. We will start using them when we write code in an IDE like Eclipse at a later stage. However, we will start using the import statements to import appropriate classes from the packages.

The import statements should occur below the package statement and above the class declaration in the source code.

Select all the correct statements from the below examples:

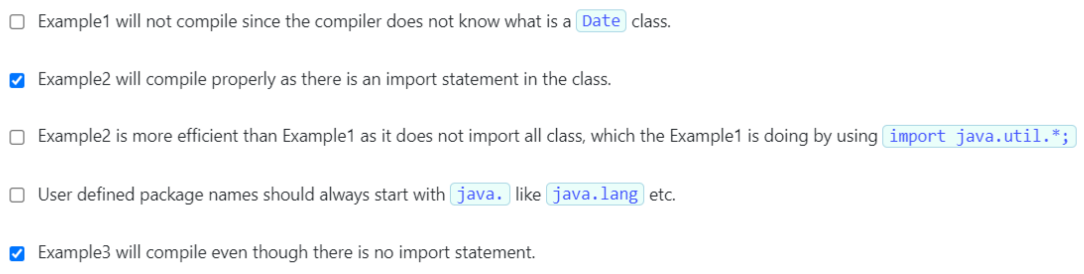
```
import java.util.*;


public class Example1 {
        public static void main(String[] args) {
                Date currentDate = new Date();
                System.out.println(currentDate);
        }
}



import java.util.Date;
public class Example2 {
        public static void main(String[] args) {
                Date currentDate = new Date();
                System.out.println(currentDate);
        }
}
```

```
public class Example3 {
        public static void main(String[] args) {
                java.util.Date currentDate = new java.util.Date();
                System.out.println(currentDate);
        }
}
```



☐ Example1 will not compile since the compiler does not know what is a `Date` class.

[✓] Example2 will compile properly as there is an import statement in the class.

☐ Example2 is more efficient than Example1 as it does not import all class, which the Example1 is doing by using `import java.util.*;`

☐ User defined package names should always start with `java.` like `java.lang` etc.

[✓] Example3 will compile even though there is no import statement.

## Q2. Encapsulation meaning and usage

Encapsulation is a feature in object-oriented languages, like Java where we can selectively hide the data and members, so that they are not accidentally corrupted by code.

For example in the below class:

```
public class Person {
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }

        private void setName(String name) {
                this.name = name;
        }

        private void setAge(int age) {
                if (age > 0) {
                        this.age = age;
                }
                //we can also flag an error when invalid age is passed}
}
```

String name and int age are declared with private access modifier.

private modifier indicates that those members are accessible only by code of that class. They are not visible to other classes.

For example, in the below code on line 36 Test class is accessing the private field name of Person class.

The compiler will give an error saying that the field name field in Person is not visible to Test.

Click on the Submit button to see the error.

If the fields name and age were marked as public when they are declared in Person, compiler would not flag an error. In such a case the below code when written in the main method of Test class would not give a compilation error.

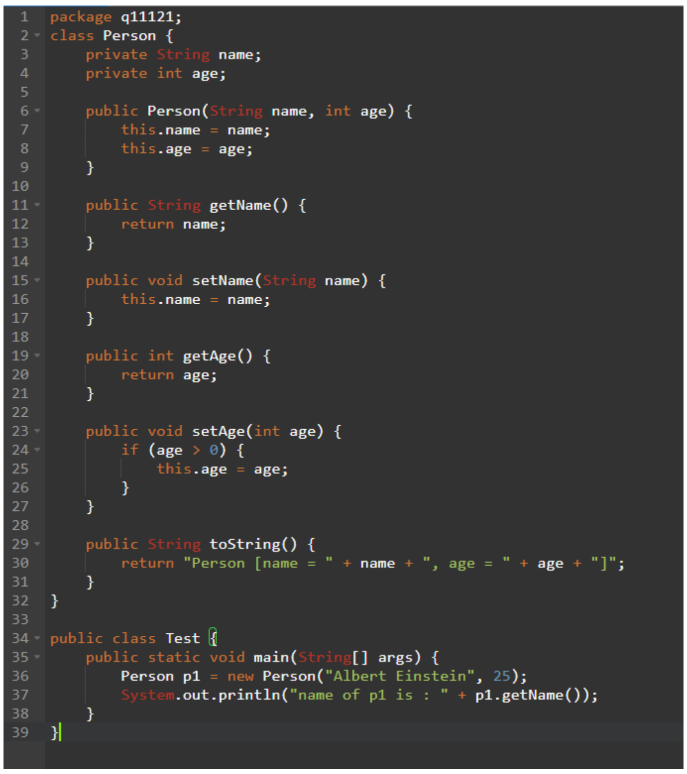
```
p1.name = -3;
```

However, you would notice that even though the value of (-3) is invalid for age, if the field age in Person was marked as public, compiler would allow such code to be written in other classes.

In the below code since the field age was declared as private, if some other class like Test wants to update age on a person object, it is forced to use the public method setAge() and pass an value to that method like : p1.setAge(-3).

By declaring the age field as private and its accessor method as public, the Person class can perform validation checks on the value passed to its setAge() method before assigning it to the age. By this the internal data of Person class is protected from being set to inconsistent or invalid state by code in other classes.

To resolve the compilation error, delete the complete code in line 36 (containing p1.name) which is wrong as it is illegal to access private members of other classes.



```
package q11121;
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }

    public String toString() {
        return "Person [name = " + name + ", age = " + age + "]";
    }
}

public class Test {
    public static void main(String[] args) {
        Person p1 = new Person("Albert Einstein", 25);
        System.out.println("name of p1 is : " + p1.getName());
    }
}
```

Q3. Different access modifiers in Java

Access modifiers are special keywords which can be associated to members like fields, methods, classes. These access modifiers define the degree of access control to those members.

The four different access control modifiers are:

1. private - marks the member as accessible only in the declaring class
2. package/default - marks the member as accessible to classes in the same package
3. protected - marks the members accessible in subclasses of the class and to classes in the same package
4. public - marks the members as accessible to all classes

Note: When none of the access modifier keywords like public, private or protected are used, the default or package access restrictions apply.

Select all the correct statements for the below code:

```
class A {
        private int x;
        public int y;
        public void sayHello() {
                System.out.println("Hello");
        }
        private void sayBye() {
                System.out.println("Hello");
        }
}
class B {
        public static void main(String[] args) {
                A a1 = new A();
                System.out.println(a1.x); // Statement 1
                System.out.println(a1.y); // Statement 2
                System.out.println(a1.sayHello());  // Statement 3
                System.out.println(a1.sayBye()); // Statement 4
        }
}
```

☐ Statement 1 will not produce a compilation error.

☐ Statement 2 will produce a compilation error.

[✓ Statement 3 will not produce a compilation error.]

☐ Statement 4 will not produce a compilation error.

## Different types of Variables

Q1. Understanding different types of variables

In Java, we can classify variables into two types:

1. primitive variables - these are also simply called as variables
2. references

Below are examples of variables ( formally called as primitive variables):

```
int x = 3;
char gender = 'F';
boolean isCorrectCode = true;
```

In the above declarations, you will notice that the identifiers x, gender and isCorrectCode are of primitive integer (int), character (char) and boolean data types respectively.

Henceforth we will use the word variable to refer to variables whose type is of primitive type (int, short, long, float, byte, double, char and boolean).

Below are examples of references ( formally called as reference variables):

```
String firstName = "Appolonius";
Date currentDate = new Date();
Student st1 = new Student("1007", "Ganga", 25, 'F'); //assuming there is a custom class called Student with the constructorint[] luckyNumbersArr = {3, 5, 7};
String[] cityNamesArr = {"Jerusalem", "Varanasi", "Rome"};
```

Hereafter we will refer to all variables of class types or interface types or even array types as references.

We can assume a variable as a placeholder that stores a value. For example, in the above declarations variable x is like a cup that stores a value 3.

Similarly we can assume a reference as a placeholder (cup) that points to the object.

In Java, instances of arrays (can be primitive array like int[] or a array of a class like String[]) and instances of classes are objects. We can use the dot . to access a member of that object.
For example:

```
String firstName = "Appolonius";
Student st1 = new Student("1007", "Ganga", 25, 'F'); //assuming there is a custom class called Student with the constructorint[] luckyNumbersArr = {3, 5, 7};
System.out.println(firstName.length()); // prints 10 System.out.println(st1.getId()); // prints 1007System.out.println(luckyNumbersArr.length); // prints 3
```

Select all the correct statements for the below code:

```
int age = 7;
Student[] studentArr = new Student[10];
long currentTimeInMilliSeconds = System.currentTimeMillis();
String welcomeMsg = "Hello, welcome to Java Gym!"
boolean[] flagsArr = new boolean[10];
```

- [ ] `age` is called a `reference`
- [ ] The line containing `studentArr` will result in a compilation error, since we cannot create an array of type Student class.
- [✓ `currentTimeInMilliSeconds` is a variable and not reference.]
- [ ] `studentArr` is not a reference.
- [✓ `welcomeMsg` is a reference.]
- [✓ `flagsArr` is a reference.]

## Void

### Q1. Usage of void

In Java, the void keyword is a placeholder to indicate that there is no return type for a method.

For example, the below method adds num1 and num2 and returns the result.

```
public int sum(int num1, int num2) {
        return num1 + num2;
}
```

Since the type of the result being returned is also int, the method's return type is marked int.

In the below example, you will notice that the method printSum does not return but prints the summation to the System's console.

```
public void printSum(int num1, int num2) {
        System.out.println( num1 + num2);
}
```

Since the method printSum does not return any value, the return type of the method is void. By this the compiler understands that this method does not return any.

Considering the above two methods are available, select all the correct statements from the below code:

```
int total1 = printSum(4, 5); //Statement 1
int total2 = sum(4, 5); //Statement 2
System.out.println(printSum(1, 2)); //Statement 3
System.out.println(sum(1, 2)); //Statement 4
```

- [ ] In Statement 1, the value of `total1 = 9`.
- [✓ In Statement 2, the value of `total2 = 9`.]
- [ ] Statement 3 prints a value of 3 to console.
- [✓ Statement 4 prints a value of 3 to console.]