Lab 03: Classes and Objects

Objective(s):

1. What is Class?
2. What is an Object?
3. Class vs Object
4. Assigning Object References Variables
5. Method in class
6. Constructor
7. The this Keyword
8. Stack Class

1: What is Class?

Class:

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint/prototype or a set of instruction to build a specific type of object or blueprint that defines a nature of a future object.

Syntax:

```
class <class_name>{
    field;
    method;
  }
```

2: What is an Object?

An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You could also expect a person to be able to do something, such as walking or driving. This would be considered a method of the person.
Or in other words, an object is a self-contained component which consists of methods and properties to make a particular type of data useful. Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

An instance is a specific object created from a particular class. Classes are used to create and manage new objects.

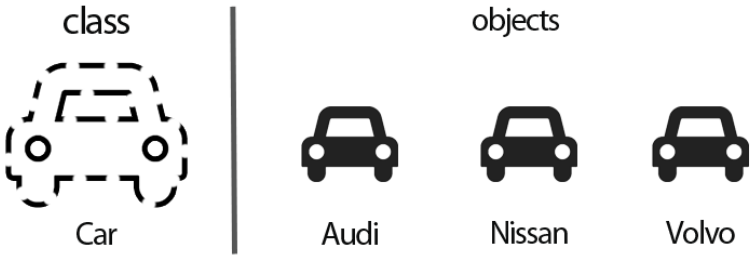
Syntax:

```
ClassName ReferenceVariable = new ClassName();
```

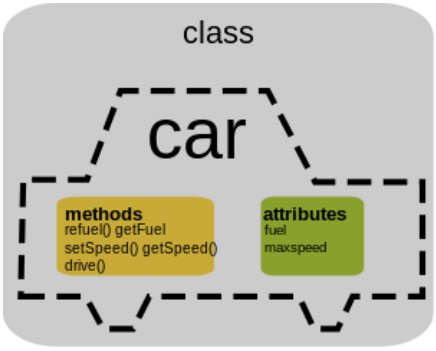
Example:

```
Car car; //declare reference to object
car = new Car(); //allocate a car object
```

3: Class vs Object



Class containing attributes and methods.



4: Assigning Object References Variables:

- We can assign value of reference variable to another reference variable.
- Reference Variable is used to store the address of the variable.
- Assigning Reference will not create distinct copies of Objects.
- All reference variables are referring to same Object.

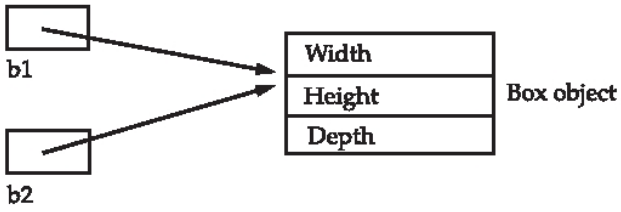
Assigning Object Reference Variables does not:

1. Create Distinct Objects.
2. Allocate Memory.

3. Create duplicate Copy.

Consider below example:

```
Box b1 = new Box();
Box b2 = b1;
```



- b1 is reference variable which contain the address of Actual Box Object.
- b2 is another reference variable
- b2 is initialized with b1 means – "b1 and b2" both are referring same object , thus it does not create duplicate object , nor does it allocate extra memory

5: Method in class

A method is an action that an object is able to perform. Consider the example of an object of the type '**person**' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

A function is a **combination of instructions** that are combined to achieve some result.

Syntax:

```
type name(parameter_list){
   //body of method
}
```

6: Constructor

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

A house needs a Builder and an object need a constructor in JAVA. For creating a new object/instance of a class you need a constructor. 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

Note that the constructor name must match the class name, and it cannot have a return type (like void).

Also note that the constructor is called when the object is created.

If you do not create a constructor, one will be created for you. This is called the **default constructor**.

Syntax:

```java
public class MyClass {
  int x;
  // Constructor with a parameter
  public MyClass(int x) {
    this.x = x;
  }
  // Call the constructor
  public static void main(String[] args) {
    MyClass myObj = new MyClass(5);
    System.out.println("Value of x = " + myObj.x);
  }
}
```

7: The this Keyword

Sometimes a method or constructor will need to refer to the object that invoked it. To allow this, JAVA defines the **this** keyword, **this** can be used inside any method to refer to the *current object*. That is, **this** is always a reference to the object on which the method was invoked.

The most common use of the **this** keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

```java
public class MyClass {
  int x;
  // Constructor with a parameter
  public MyClass(int x) {
    this.x = x;
  }
}
```

8: Stack Class

A stack stores data using first-in, last-out ordering. That is, a stack is like a stack of plates on a table—the first plate put down on the table is the last plate to be used. Stacks are controlled through two operations traditionally called push and pop. To put an item on top of the stack, you will use push. To take an item off the stack, you will use pop. As you will see, it is easy to encapsulate the entire stack mechanism.

Here is a class called Stack that implements a stack for up to ten integers:

```
// This class defines an integer stack that can hold 10 values
class Stack {
  int stck[] = new int[10];
  int tos;

  // Initialize top-of-stack
  Stack() {
    tos = -1;
  }

  // Push an item onto the stack
  void push(int item) {
    if(tos==9)
      System.out.println("Stack is full.");
    else
      stck[++tos] = item;
  }

  // Pop an item from the stack
  int pop() {
    if(tos < 0) {
       System.out.println("Stack underflow.");
       return 0;
    }
    else
      return stck[tos--];
  }
}
```

The class TestStack, shown here, demonstrates the Stack class. It creates two integer stacks, pushes some values onto each, and then pops them off.

```
class TestStack {
  public static void main(String args[]) {
    Stack mystack1 = new Stack();
    Stack mystack2 = new Stack();

    // push some numbers onto the stack
    for(int i=0; i<10; i++) mystack1.push(i);
    for(int i=10; i<20; i++) mystack2.push(i);

    // pop those numbers off the stack
    System.out.println("Stack in mystack1:");
    for(int i=0; i<10; i++)
       System.out.println(mystack1.pop());

    System.out.println("Stack in mystack2:");
    for(int i=0; i<10; i++)
       System.out.println(mystack2.pop());
  }
}
```

This program generates the following output:

Stack in mystack1:	Stack in mystack2:
9	19
8	18
7	17
6	16
5	15
4	14
3	13
2	12
1	11
0	10

Lab Tasks:

Exercise 1(a) — Car.java

Implement a class Car, that has the following characteristics:

a) brandName,
b) priceNew, which represents the price of the car when it was new,
c) color

d) odometer, which is milo meter shows number of milage travelled by car

The class should have:

A. A method getPriceAfterUse() which should return the price of the car after being used according to the following formula:

$$car\ price\ after\ being\ used = priceNew * (1 - \frac{odometer}{600000})$$

B. A method updateMileage(double traveledDistance) that changes the current state of the car by increasing its milage,

C. A method outputDetails() that will output to the screen all the information of the car, i.e., brand name, price new, price used, color, and odometer.

Exercise 1(b) — TestCar.java

Write a test class for the Car class above. You are required to do the followings:

a. Create an object of type Car.
b. Assign any valid values to the instance variables of the object created in 'A'.
c. Use the method getPriceAfterUse on the object created in 'A' then output the result to the screen.
d. Use the method updateMilage on the object created in 'A' by passing a valid value.
e. Do part 'C' again.
f. Use the method outputDetails on the object created in 'A'.

Exercise 2(a) — Flight.java

Design then implement a class to represent a **Flight**. A Flight has a *flight number*, a *source*, a *destination* and a *number of available seats*. The class should have:

a. A **constructor** to initialize the 4 instance variables. You have to shorten the name of the source and the destination to 3 characters only if it is longer than 3 characters by a call to the method in the 'j' part.

b. An **overloaded constructor** to initialize the *flight number* and the *number of available seats* instance variables only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string, i.e." ")

c. An **overloaded constructor** to initialize the *flight number* instance variable only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string; and the *number of available seats* to zero)

d. One **accessor/getter** method for each one of the 4 instance variables.

e. One **mutator/setter** method for each one of the 4 instance variables **except** the *flight number* instance variable.

f. A method `public void reserve (int numberOfSeats)` to reserve seats on the flight.

(**NOTE:** You have to check that there is enough number of seats to reserve)

g. A method `public void cancel(int numberOfSeats)` to cancel one or more reservations

h. A `toString` method to easily return the flight information as follows:

```
Flight No: 1234
From: KAR
To: LAH
```

i. An `equals` method to compare 2 flights.

(**NOTE:** 2 Flights considered being equal if they have the same flight number)

j. The following method:

```
private String shortAndCapital (String name) {
  if (name.length() <= 3) {
    return name.toUpperCase();
  } else {
    return name.substring(0,3).toUpperCase();
  }
}
```

Exercise 2(b) — FlightTest.java

Write a test class for the Flight class you wrote. You should try to use all the methods you wrote.

Exercise 3(a) — Ship.java

MyJava Coffee Outlet runs a catalog business. It sells only one type of coffee beans. The company sells the coffee in 2-lb bags only and the price of a single 2-lb bag is $5.50. When a customer places an order, the company ships the order in boxes. The boxes come in 3 sizes with 3 different costs:

	Large box	Medium box	Small box
capacity	20 bags	10 bags	5 bags
cost	$1.80	$1.00	$0.60

The order is shipped using the least number boxes. For example, the order of 52 bags will be shipped in 2 boxes: 2 large boxes, 1medium and 1 small.

Exercise 3(b) — ShipTest.java

Develop an application that computes the total cost of an order.

Sample output:

```
Number of Bags Ordered: 52
The Cost of Order: $ 286.00

Boxes Used:
        2 Large  - $3.60
        1 Medium - $1.00
        1 Small  - $0.60

Your total cost is: $ 291.20
```

Exercise 4 — Phone.java

A cell phone was created to provide basic functionality of Calling and Messaging. With the course of time, thousands of new features have been added and the count is still increasing but the basic features remain same. For any cell phone object, design a class that describes its basic characteristics and methods.

Remember, every cell phone has:

An IMEI code

A SIM card

A Processor

Internal Memory

Single SIM/ Dual SIM

Every cell phone can:

Dial a Number

Receive a Call

Send an SMS

Receive an SMS

Create parameterized constructor and getters and setters for all the instance variables.

Create an application to test your class that has an object cellPhone1 whose IMEI code is: **`IEDF34343435235,`** accepts a Nano SIM card, with SnapDragon as its processor, has 8 GB internal memory and is Single SIM. Call all the methods of cellPhone1.

Exercise 5 — Date.java

Create your version of **`Date`** class that includes three instance variables—a **`month`** (type int), a **`day`** (type int) and a **`year`** (type int). Provide a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a *set* and a *get* method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test app named DateTest that demonstrates class Date's capabilities.

END