



Mohammad Ali Jinnah University Karachi

Department of Computer Science

CS1421: Object Oriented Programming LAB 04

Instructors

Safiyah Batool

Samia Bashir

Lab 4

Classes, Objects, Methods and Constructors

Objective

After completing this lab, the students should be able to

- Create methods
- Define classes and create objects
- Access objects via object reference variables
- Understand the usage of constructors
- Define private data fields with appropriate getters and setters (Encapsulation)

Introduction

The class is a logical construct that defines the shape and nature of an object .As it is the prime unit of execution for object oriented programming in java, so any concept in java program must be encapsulated within the class.

In java, class is defined as a new data type. This data type is used to create objects of its type. Each object created from the class contains its own copy of the attributes define in the class. The attributes are also referred to as fields and represent the state of an object. The initialization of objects is done using constructors and the behavior of the objects is defined using methods.

Declaring a class

As class declaration should begin with the keyword class followed by the name of the class that is being declared.

Besides this, following are some conventions to be followed while naming a class:

- Class name should be a noun.
- Class name can be in mixed case, with the first letter of each internal word capitalized.
- Class name should be simple, descriptive, and meaningful
- Class name cannot be java keywords
- Class name cannot begin with a digit . however they can begin with a dollar(\$) symbol or an underscore character:

Syntax:

```
Class <class_name>{  
    //class body  
}
```

Example:

```
public class Person {  
    /*  
    Person's properties(fields) and  
    behavior(methods) will be  
    defined here.  
    */  
}
```

Access Modifiers:

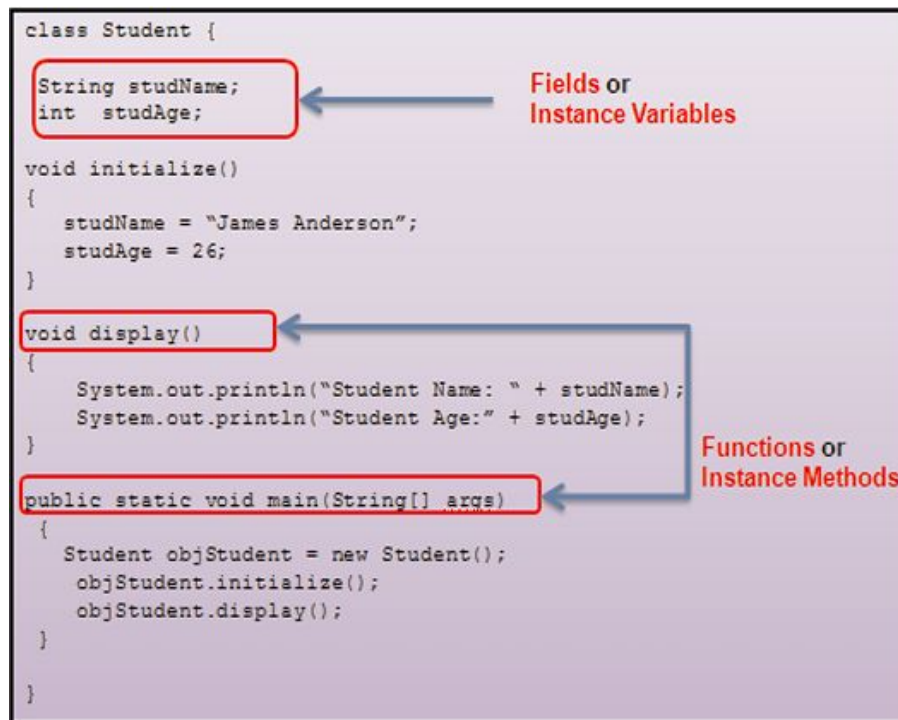
A *Java access modifier* specifies which classes can access a given class and its fields, constructors and methods. Access modifiers can be specified separately for a class, its constructors, fields and methods.

- public
- private
- default (package)
- protected

Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

The following figure shows the declaration of a sample class.



Creating objects

Objects are the actual instances of the class.

Declaring and Creating an Object:

An object is created using the `new` keyword. On encountering the `new` keyword JVM allocates memory for the object and returns a reference or memory address of the allocated object. The reference or memory address is then stored in a variable. This variable is also called as reference variable.

The syntax for creating an object is as follows

Syntax:

```
<class_name> <object_name> = new <class_name> ();
```

Member of a class

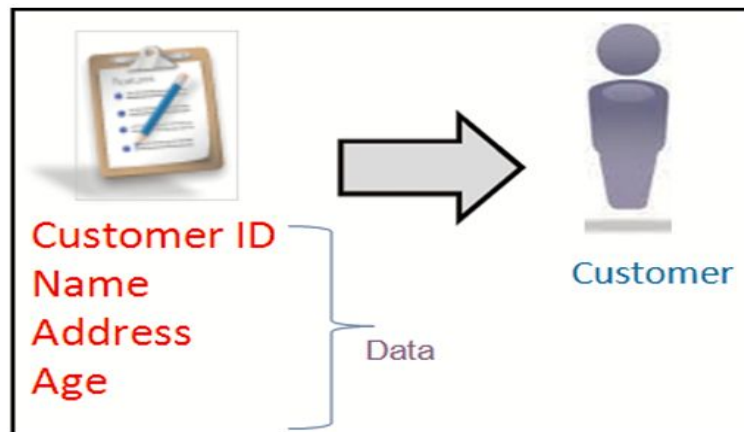
The member of a class are fields and methods. Fields define the state of an object created from the class and are referred to as instance variable the methods are used to implement the behavior of the objects and are referred as instance methods.

Instance Variables

The fields or variables defined within a class are called instance variables .Instance variables are used to stored data in them. They are called instance variables because each instance of the class that is objects of that class will have its own copy of the instance variable. This means each object of the class will contain instance variable during creation.

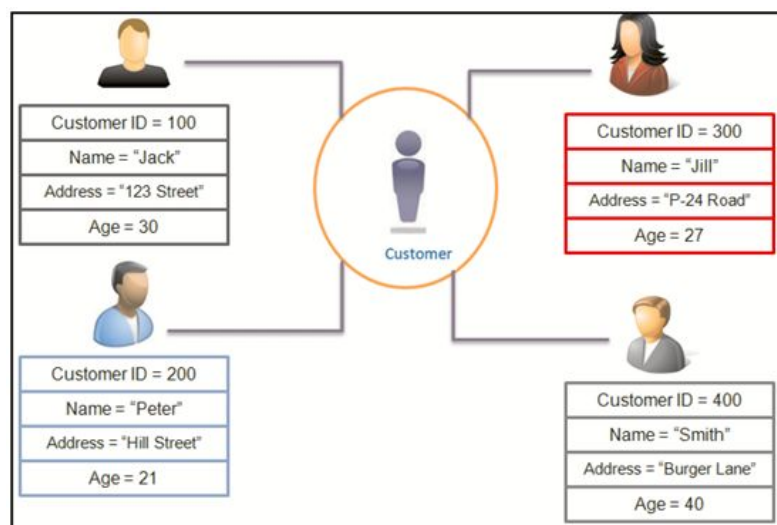
Consider a scenario where the customer class represent the details of customers holding accounts in a bank. In this scenario a typical question that can be asked is ‘What are the different data that are required to identify a customer in a banking domain and represent it as a single object?’

The following shows a customer object with its data requirement.



As shown in figure the identified data requirements for a bank customer includes : CustomerID, Name, Address, and Age. To map these data requirements in a customer class, instance variables are declared. Each instances created from the customer class will have its own copy of the instances variables

Shows figure various instances of the class with its own copy of instance variables



As shown in above figure each instance of the class has its own instance variables initialized with unique data. Any changes made to the instance variables of one object will not affect the instance variables of the another object

The syntax to declare an instance variable within a class is as follows:

Syntax:

```
[access_modifier] data_type instanceVariableName;
```

Where

- **access_modifier** is an optional keyword specifying the access level of an instance variable. It could be private, protected, and public.
- **data_type** specifies the data type of the variable. It can be of primitive types such as int, float, boolean, and so on. Also it can be of reference types, such as strings, arrays, or objects.
- **instanceVariableName** specifies the name of the variable.

```
public class Customers {
    //Declare instance Variables
    int customerAge;
    String customerName;
    String customerAddress;

    /* As main() method is the member of this class so
       it can access other members of the class */
    public static void main(String args[])
    {
        //Declare and instantiate objects of type Customers
        Customers customer1=new Customers();
        Customers customer2=new Customers();

        //Access the instance variables to store information of Customer1
        customer1.customerName="John";
        customer1.customerAge=14;
        customer1.customerAddress="3-73 Street";

        //Access the instance variables to store information of Customer2
        customer2.customerName="Jane";
        customer2.customerAge=28;
        customer2.customerAddress="422 street";

        //Display the information of object customer1
        System.out.println("Customer 1 Name:"+customer1.customerName);
        System.out.println("Customer 1 Age: "+customer1.customerAge);
        System.out.println("Customer 1 Address:"+customer1.customerAddress);

        //Display the information of object customer2
        System.out.println("Customer 2 Name:"+customer2.customerName);
        System.out.println("Customer 2 Age: "+customer2.customerAge);
        System.out.println("Customer 2 Address: "+customer2.customerAddress);

    }
}
```

Instance Methods

They are functions declared in a class. They implement the behavior of an object. They are used to perform operations on the instance variables. They can be accessed by instantiating an object of the class in which it is defined and then, invoking the method. Instance method can access instance variables and instance methods directly.

For example, the class Car can have a method a Brake() that represents the 'Apply Brake' action. To perform the action, the method Brake() will have to be invoked by an object of class Car.

Following conventions have to be followed while naming a method:

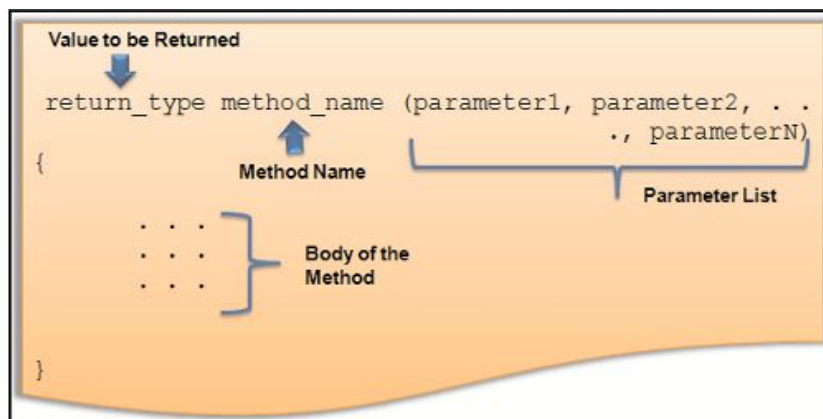
- Cannot be a Java keyword.
- Cannot contain spaces.
- Cannot begin with a digit.
- Can begin with a letter, underscore, or a '\$' symbol.
- Should be a verb in lowercase.
- Should be descriptive and meaningful.

Syntax:

```
[access_modifier] <return type> <method_name> ([list of parameters]) {  
  
    // Body of the method  
  
}
```

where

1. Modifiers — such as public, private, protected and default.
2. The return type — the data type of the value returned by the method, or void, if the method does not return a value.
3. The method name — the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis — a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses (). If there are no parameters, you must use empty parentheses.
5. The method body, enclosed between braces — the method's code, including the declaration of local variables.



Each instance of the class has its own instance variables, but the instance methods are shared by all the instances of the class during execution. As methods are part of class declaration, they can access the other class member, such as instance variables and method of the class.

Invoking Methods

You can access a method of a class by creating an object of the class. To invoke a method, the object name is followed by the dot operator (.) and the method name. A method is always invoked from another method. The method which invokes a method is referred to as the calling method. The invoked method is referred to as the **called method**. After execution of all the statements within the code block of the invoked method, the control returns back to the **calling method**.

Most of the methods are invoked from the main() method of the class which is the entry point of the program execution.

When a program invokes a method, the program control gets transferred to the called methods. When the method is invoked, all the statements that are part of the method would be executed. When the JVM invokes a class method, it selects the method to invoke based on the type of object reference which is always known as compile time.

code snippet that demonstrates a class with main () method which creates the instance of the class **Customer** and invokes the methods defined in the class

```
public class Customers {
    //Declare instance Variables
    int customerAge;
    String customerName;
    String customerAddress;

    //Instance Method to display object Information
    public void display(){
        System.out.println("Customer Name:"+customerName);
        System.out.println("Customer Age: "+customerAge);
        System.out.println("Customer Address: "+customerAddress);
    }
    /* As main() method is the member of this class so
       it can access other members of the class */
    public static void main(String args[])
    {
        //Declare and instantiate an object of type Customers
        Customers customer1=new Customers();
        Customers customer2=new Customers();

        customer1.customerName="John";
        customer1.customerAge=14;
        customer1.customerAddress="3-73 Street";

        customer2.customerName="Jane";
        customer2.customerAge=28;
        customer2.customerAddress="422 street";
    }
}
```



```

customer1.display();
customer2.display();
customer2.customerAddress="550 street";
customer2.display();
    }
}

```

Another Example:

```

public class Calculator {
    public int add(int n1,int n2){
        return n1+n2;
    }

    public int sub(int n1,int n2){
        return n1-n2;
    }
    public int mul(int n1,int n2) {
        return n1*n2;
    }
    public float div(float n1,float n2){
        return ((float) (n1/n2));
    }

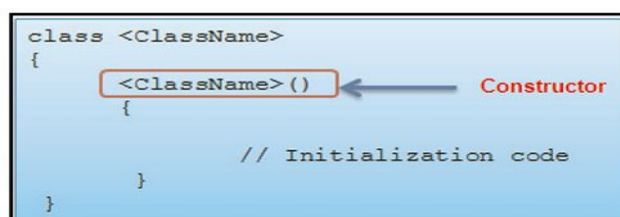
    public static void main(String[] args) {
        Calculator c= new Calculator();
        System.out.println("Addition: "+c.add(2,3));
        System.out.println("Subtraction: "+c.sub(4,2));
        System.out.println("Multiplication: "+c.mul(6,3));
        System.out.println("Division: "+c.div(5,3));
    }
}

```

Constructor

A class can contain multiple variables whose declarations and initialization becomes difficult to track if they are alone within different blocks. Likewise, they may be other startup operations that need to be performed in an application like opening a file and so forth. Java programming language allows objects to initialize themselves immediately upon their creation. The behavior is achieved by defining constructors in the class.

A constructor is a method having the same name as that of the class. Constructors initialize the variables of a class or perform startup operations only once when the object of the class is instantiate. They are automatically executed whenever an instance of a class is created before the new keyword completes. Also, constructor method do not have return types, but accepts parameters



```

class <ClassName>
{
    <ClassName>() ← Constructor
    {
        // Initialization code
    }
}

```

Declaration of the constructor

Syntax:

```
<classname>() {  
  
    // Initialization code  
  
}
```

Code snippet

```
public class Rectangle {  
    int width;  
    int height;  
    /**  
     * Constructor for Rectangle class  
     */  
    Rectangle() {  
        width = 10;  
        height = 10;  
    }  
}
```

The code declares a method name `Rectangle()` which is a constructor. This method is invoked by JVM to initialize the two instance variables, `width` and `height`, when the object of type `Rectangle` is constructed. Also, the constructor does not have any parameters; hence it is called as no-argument constructor.

Example:

A simple Example to declare a class with fields and methods

```
public class Person {  
  
    //fields/Attributes/properties  
    private String personName;  
    private int personAge;  
  
    Person()  
    {  
        personName = "Ali";  
        personAge = 10;  
    }  
    //Methods  
    void display()  
    {
```

```
        System.out.println("Name: "+personName);
        System.out.println("Age: "+personAge);
    }
}
```

Test Class:

```
public class Test {
    public static void main(String[] args) {
        Person p=new Person();
        p.display();
    }
}
```

Example: (Parameterized Constructor)

```
public class Person {

    //fields/Attributes/properties
    private String personName;
    private int personAge;

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

    public void display()
    {
        System.out.println("Name: "+ personName);
        System.out.println("Age: "+personAge);
    }
}
```

```
public class TestPerson {
    public static void main(String[] args) {
        Person p=new Person("David",32);
        p.display();
    }
}
```

```
}
```

Default Constructor:

- Created for the classes where explicit constructors are not defined.
- Initializes the instance variables of the newly created object to their default values.

Following table which lists the default values assigned to instance variables of the class depending on their data types.

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0
char	'\u0000'
boolean	False
String (any object)	Null

Parameterized constructor

The parameterized constructor contains a list of parameters that initializes instance variables of an object. The value for the parameters is passed during the object creation. This means each object will be initialized with different set of values.

```
class Example{
    //Default constructor
    Example(){
        System.out.println("Default constructor");
    }
    /* Parameterized constructor with
    * two integer arguments
    */
    Example(int i, int j){
        System.out.println("constructor with Two parameters");
    }
    /* Parameterized constructor with
    * three integer arguments
    */
    Example(int i, int j, int k){
        System.out.println("constructor with Three parameters");
    }
}
```

```

/* Parameterized constructor with
 * two arguments, int and String
 */
Example(int i, String name){
    System.out.println("constructor with int and String param");
}
public static void main(String args[]){
    //This will invoke default constructor
    Example obj = new Example();

    /* This will invoke the constructor
     * with two int parameters
     */
    Example obj2 = new Example(12, 12);

    /* This will invoke the constructor
     * with three int parameters
     */
    Example obj3 = new Example(1, 2, 13);

    /* This will invoke the constructor
     * with int and String parameters
     */
    Example obj4 = new Example(1,"BeginnersBook");
}
}

```

Parameterized constructor – A weird compilation error

While discussing default constructor, we have seen that when we do not create a default constructor, Java compiler inserts the default constructor into the code on our behalf. However this is not always true.

```

class Example{
    Example(int i, int j){
        System.out.print("parameterized constructor");
    }
    Example(int i, int j, int k){
        System.out.print("parameterized constructor");
    }
    public static void main(String args[]){
        Example obj = new Example();
    }
}

```

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

To achieve encapsulation in Java

Declare the variables of a class as private.

Provide public setter and getter methods to modify and view the variables values.

Following is an example that demonstrates how to achieve Encapsulation in Java

```
public class EncapTest {
    private String name;
    private String idNum;
    private int age;

    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getIdNum() {
        return idNum;
    }
    public void setAge( int newAge) {
        age = newAge;
    }
    public void setName(String newName) {
        name = newName;
    }
    public void setIdNum( String newId) {
        idNum = newId;
    }
}
```

The public setXyz() and getXyz() methods are the access points of the instance variables of the EncapTest class. Normally, these methods are referred as getters and setters. Therefore, any class that wants to access the variables should access them through these getters and setters.

The variables of the EncapTest class can be accessed using the following program.

```
public class RunEncap {

    public static void main(String args[]) {
        EncapTest encap = new EncapTest();
        encap.setName("James");
        encap.setAge(20);
        encap.setIdNum("12343ms");

        System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());
    }
}
```

```
}  
}
```

Lab Task:

Task 1:

Choose an object (Car, mobile). Identify its attributes/fields and behavior/methods. Create a class of that object with instance variables and instance methods. Create multiple (at least three) objects from that class, store different values in the fields and call methods for each object.

Task 2:

Create a **CustomerAccount** class with attributes **CustomerName**, **TypeOfAccount**, **AccountNumber**, **CurrentBalance** and methods such as **debit()**, **credit()**, **withdraw()** etc. Apply access modifiers to enhance security so that customer cannot directly change account balance. Use getter and setter methods to view and modify balance according to debit and credit amount. Create multiple objects for customer accounts (atleast three) and store customer information. Call methods from different objects with different information. (Apply validations where required).

Task 3:

Task 2 is a sequential process for all customers with fixed debit and credit value. Enhance the application by using control statement to take input of account number and make user choose function to perform (debit, credit, withdraw). (Inputs: Acc No, Action to perform, Amount to perform action).

Task 4:

Create a class **PSLteams** with attributes such as **teamName**, **playerNames**, **Rank**, **NoOfMatchesPlayed**, **NoOfMatchesWon**, **MatchesLost** etc. Apply access modifiers so that

- fields can not be changed but can be viewed.
- methods can be accessed from anywhere.

Design a constructor to add information of team etc. name, player names and rank, set matches played, won and lost to 0. Design method **playmatch()** which takes a single parameter '**status**' which can be **true** or **false**, where true indicates winning and false means that team has lost the match. No. of matches played, match won and lost should automatically be maintained whenever there is a match played. User will be able to view updated information at any time.

Design a class **PSLteams2019** as test class. Create objects for all PSLteams (KarachiKings, MultanSultan, IslamabadUnited, QuettaGiliadators and PeshawarZalmi). Call method **playmatch()** by every team and pass status. Check no. of matches played, won and lost.

Bonus: Try to create a mechanism of auto ranking for bonus points.

Task 5:

Write a Payroll class that uses the following arrays as fields:

- **employeeId**. An array of seven integers to hold employee identification numbers. The array should be initialized with the following numbers: 5658845 4520125 7895122 8777541 8451277 1302850 7580489
- **hours**. An array of seven integers to hold the number of hours worked by each employee
- **payRate**. An array of seven doubles to hold each employee's hourly pay rate
- **wages**. An array of seven doubles to hold each employee's gross wages. The class should relate the data in each array through the subscripts. For example, the number in element 0 of the hours array should be the number of hours worked by the employee whose identification number is stored in element 0 of the employeeId array. That same employee's pay rate should be stored in element 0 of the payRate array. In

addition to the appropriate accessor and mutator methods, the class should have a method that accepts an employee's identification number as an argument and returns the gross pay for that employee.

Evaluation Tasks

Task 1

Write a class named `RetailItem` that holds data about an item in a retail store. The class should have the following fields:

- `description`. The `description` field references a `String` object that holds a brief description of the item.
- `unitsOnHand`. The `unitsOnHand` field is an `int` variable that holds the number of units currently in inventory.
- `price`. The `price` field is a `double` that holds the item's retail price.

Write a constructor that accepts arguments for each field, appropriate mutator methods that store values in these fields, and accessor methods that return the values in these fields. Once you have written the class, write a separate program that creates three `RetailItem` objects and stores the following data in them:

	Description	Units on Hand	Price
Item#1	Jacket	12	5990.95
Item #2	Designer Jeans	40	3432.95
Item #3	Shirt	20	2494.95

Task 2

Design a class `Cannonball` to model a cannonball that is fired into the air.

A ball has

- An `x`- and a `y`-position.
- An `x`- and a `y`-velocity.

Supply the following methods:

- A constructor with an `x`-position (the `y`-position is initially 0)
- A method `move(double sec)` that moves the ball to the next position (First compute the distance traveled in `sec` seconds, using the current velocities, then update the `x`- and `y`-positions; then update the `y`-velocity by taking into account the gravitational acceleration of -9.81 m/s^2 ; the `x`-velocity is unchanged.)
- Methods `getX` and `getY` that get the current location of the cannonball
- A method `shoot` whose arguments are the angle `a` and initial velocity `v` (Compute the `x`-velocity as $v \cos a$ and the `y`-velocity as $v \sin a$; then keep calling `move` with a time interval of 0.1 seconds until the `y`-position is 0; call `getX` and `getY` after every move and display the position.)

Use this class in a program that prompts the user for the starting angle and the initial velocity. Then call `shoot()`.

Task 3

Driver's License Exam

The local Driver's License Office has asked you to write a program that grades the written portion of the driver's license exam. The exam has 20 multiple choice questions.

Here are the correct answers:

1. B	6. A	11. B	16. C
2. D	7. B	12. C	17. C
3. A	8. A	13. D	18. B
4. A	9. C	14. A	19. D
5. C	10. D	15. D	20. A

A student must correctly answer 15 of the 20 questions to pass the exam. Write a class named `DriverExam` that holds the correct answers to the exam in an array field. The class should also have an array field that holds the student's answers. The class should have the following methods:

- `passed`. Returns `true` if the student passed the exam, or `false` if the student failed
- `totalCorrect`. Returns the total number of correctly answered questions
- `totalIncorrect`. Returns the total number of incorrectly answered questions
- `questionsMissed`. An `int` array containing the question numbers of the questions that the student missed

Demonstrate the class in a complete program that asks the user to enter a student's answers, and then displays the results returned from the `DriverExam` class's methods.

Input Validation: Only accept the letters A, B, C, or D as answers.

Home Tasks

Task 1

Hitachi Ltd. has hired you to program the television sets they manufacture. Each TV has a **channel**, **volumeLevel**, and a **state**(on/off. The state should be true if television is on and false if it is off). In the same way, every TV set can perform a few functions that are **turnOn()** to turn the television on, **turnoff()** to switch it off, **setChannel(newChannel)** that provides a specific channel to set, **setVolume(newVolumeLevel)** to set a specified volume, **channelUp()** to go up by one channel, **channelDown()** to go one channel down, **volumeUp()** in order to increase volume level by one and **volumeDown()** to decrease volume level by one. Design a class **TV** with appropriate instance variables, methods, constructors, getters and setters according to the information given above. Write a Java app to test your TV class.

Task 2

Pakistan Stock Exchange provides a reliable, orderly, liquid and efficient digitized market place where investors can buy and sell listed companies' common stocks and other securities. They have asked you to design a class for their Stocks. Create a class **Stock**

- A **string** data field named **symbol** for the stock's symbol.
- A **string** data field named **name** for the stock's name.
- A **double** data field named **previousClosingPrice** that stores the stock price for the previous day.
- A **double** data field named **currentPrice** that stores the stock price for the current time.
- A constructor that creates a stock with the specified symbol and name.
- A method named **getChangePercent()** that returns the percentage changed from **previousClosingPrice** to **currentPrice**.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a **Stock** object with the stock symbol **ORCL**, the name **Oracle Corporation**, and the previous closing price of **34.5**. Set a new current price to **34.35** and display the price-change percentage.

Task 3

While exercising, you can use a heart-rate monitor to see that your heart rate stays within a safe range suggested by your trainers and doctors. According to the American Heart Association (AHA), the formula for calculating your maximum heart rate in beats per minute is 220 minus your age in years.

Your target heart rate is a range that's 50–85% of your maximum heart rate. [Note: These formulas are estimates provided by the AHA. Maximum and target heart rates may vary based on the health, fitness and gender of the individual. Always consult a physician or qualified health-care professional before beginning or modifying an exercise program.] Create a class called **HeartRates**. The class attributes should include the person's first name, last name and date of birth (consisting of separate attributes for the month, day and year of birth). Your class should have a constructor that receives this data as parameters.

For each attribute provide set and get methods. The class also should include a method that calculates and returns the person's age (in years), a method that calculates and returns the person's maximum heart rate and a method that calculates and returns the person's target heart rate. Write a Java app that prompts for the person's information, instantiates an object of class **HeartRates** and prints the information from that object—including the person's first name, last name and date of birth—then calculates and prints the person's age in (years), maximum heart rate and target-heart-rate range.