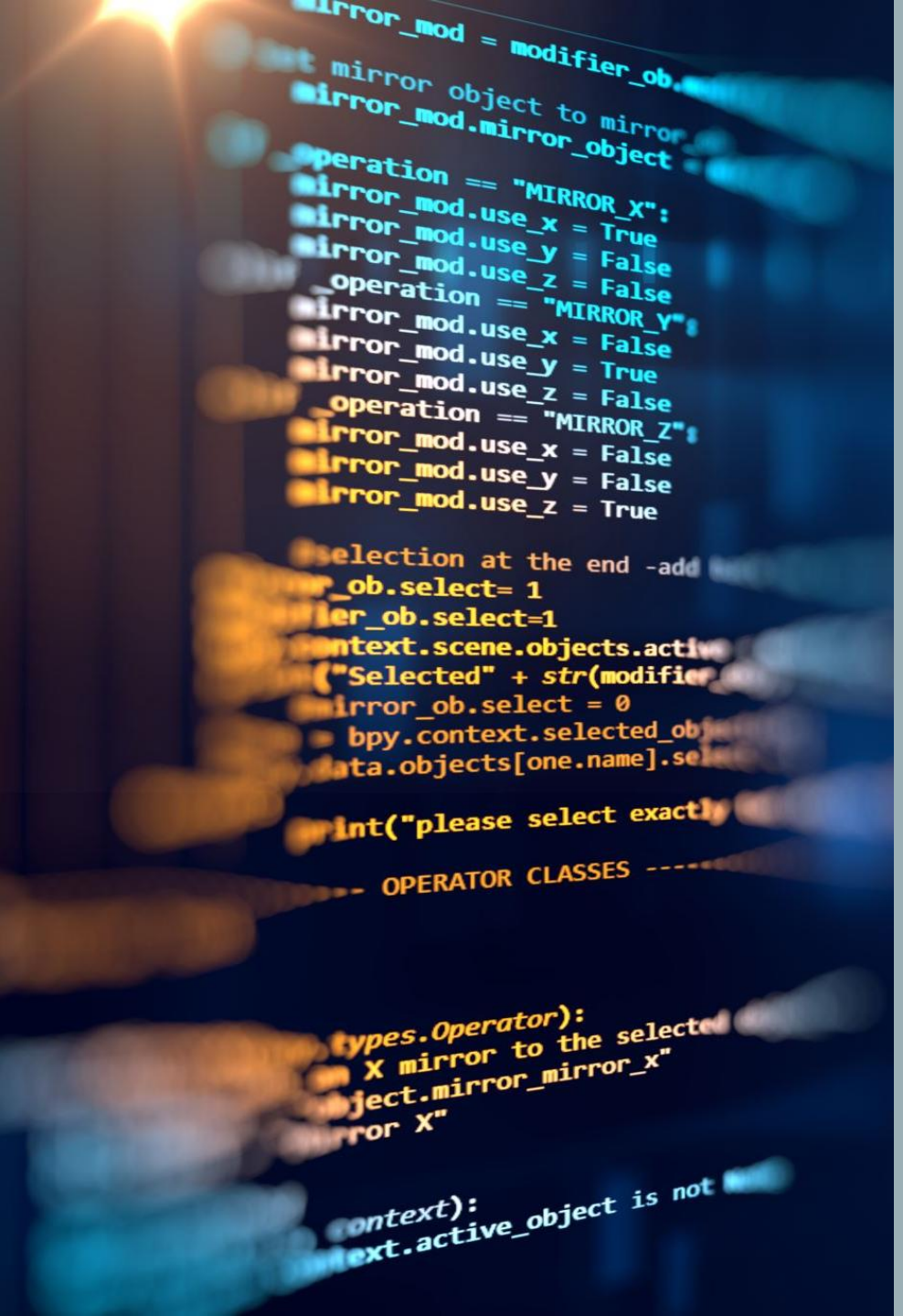


ICT 231-3 OBJECT ORIENTED PROGRAMMING

LECTURE 1 – OBJECT ORIENTED CONCEPTS

Mrs. W.B.M.S.C. Wijayakoon
Lecturer (Probationary)
B.Sc., M.Eng., D.Eng.(Reading)
Faculty of Technological Studies
Uva Wellassa University of Sri Lanka
Date: 27/09/2022

Outline

- **What is OOP?**
- **Objects and Classes:**
  - Working with objects and classes
  - Defining constructors, fields and methods
  - Declaring, initializing objects
  - Working with object references

Object Oriented Programming

- A methodology or paradigm to design a program using classes and objects
- Objects are the most important part of a program
- Object means a real-world entity
- Goal of OOP
  - Manipulating objects to get results
- OOP allows user to create objects that they want and then create methods to handle those objects

OOP Concepts

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Java Naming Conventions

- Makes code easier to read
- Suggested by Java Communities such as Sun Micro Systems and Netscape
- Some conventions must be followed by every identifier
  - Name must not contain any white spaces
  - Name should not start with a special character
  - Name should follow camel-case
  - Use appropriate words instead of acronyms

Java Naming Conventions

- Class
  - Should start with the uppercase letter
  - Should be a noun

```
public class Employee
{
//code snippet
}
```

Java Naming Conventions

- Interface
  - Should start with the uppercase letter
  - Should be a noun

```
interface Printable
{
//code snippet
}
```

Java Naming Conventions

- Variable
  - Should start with the lowercase letter
  - Should not start with the special characters
  - If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter
  - Avoid using one-character variables

```
class Employee
{
//variable
int id;
//code snippet
}
```

Java Naming Conventions

- Package
  - Should start with the lowercase letter
  - If the name contains multiple words, separate by dots.

```
package com.javatpoint; //package
class Employee
{
//code snippet
}
```

Java Naming Conventions

- Constant
  - Should be in uppercase letters
  - If the name contains multiple words, separate by an underscore.
  - May contain digits but not as the first letter.

```
class Employee
{
//constant
 static final int MIN_AGE = 18;
//code snippet
}
```

Object

- Represents an entity in the real world that can be distinctly identified
- Can be
  - Physical - Car, Student
  - Logical - Loan
- An object has a unique identity, state and behavior

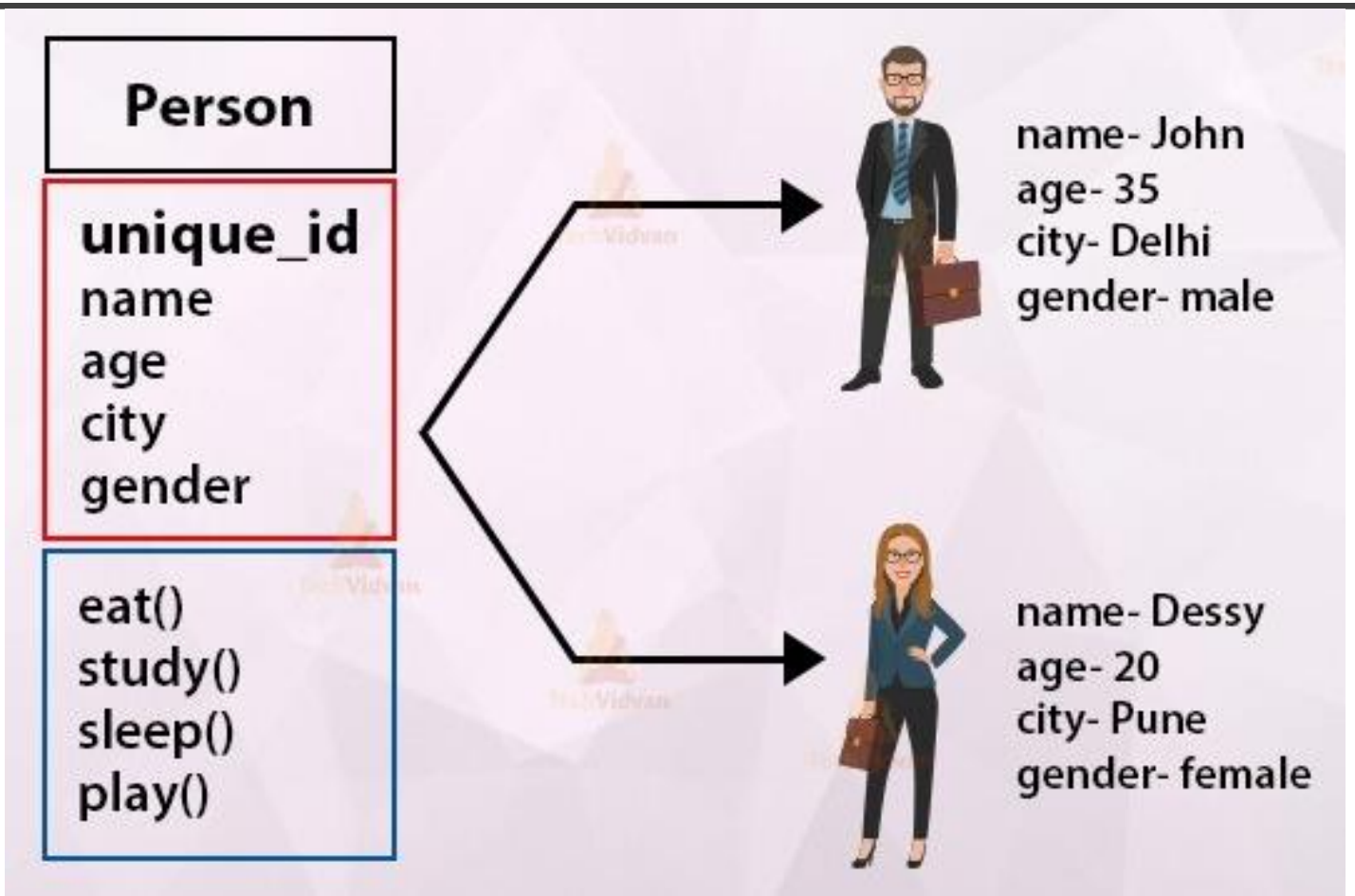
Object

- State (Properties/Attributes) of an object is represented by data fields with their current values.
  - Object - Student
  - Attributes - name, age, school, grade
- Behaviour of an object is represented by methods.
  - Object - Car
  - Methods - getSpeed()
- Object is performing an action when invoking a method on an object.

Class

- A class is a template or a blueprint that defines the form (attributes and methods) of an object.
- A set of plans that specify how to build an object.
- Objects are instances of a class. You can create multiple instances of a class
- Java uses a class specification to construct objects

Example


Person
unique_id
name
age
city
gender
eat()
study()
sleep()
play()
name- John
age- 35
city- Delhi
gender- male
name- Dessy
age- 20
city- Pune
gender- female

Class

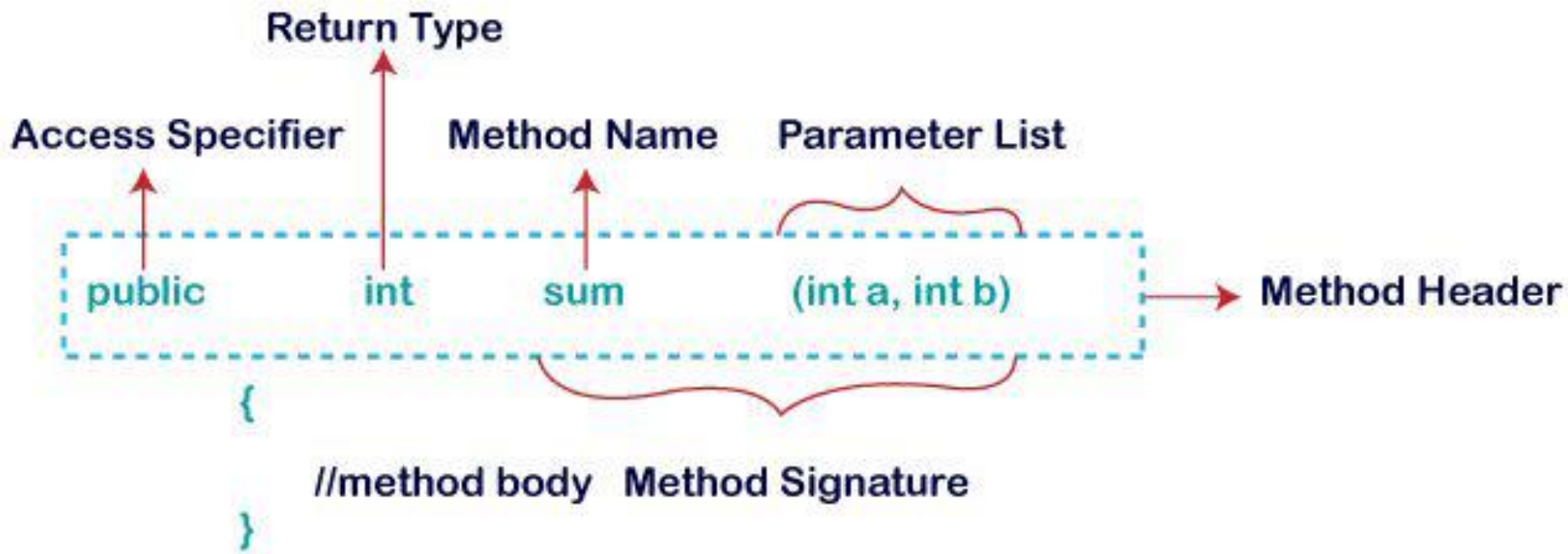
- A class is created by using the keyword class
- A java class uses variables to define attributes and methods to define actions
- Attributes are also referred to as instance variables
- A class provides a special; type of methods known as **constructors** which are invoked to create a new object
- A constrictor is designed to perform initialization actions

Methods

- A method is a block of code or collection of statements to perform a certain task
- Can write a method once and use it many times
- The method is executed only when we call or invoke it
- Methods are used to achieve the reusability of code
- Helps in easy modification
- Increase readability
- Most important method is main() method

Methods

Method Declaration:



Methods

- **Access specifier** specifies the visibility of the method. Java provides 4 types of access specifier as **public**, **private**, **protected** and **default**.
  - **Private**: available to member functions of the class
  - **Protected**: available to member functions and derived classes
  - **Public**: available to entire world
- The **default** member accessibility is **private**, meaning that only class members can access the data member or member function.

Methods

- **Return Type** is a data type of the value that the method returns. If the method does not return anything, use **void** keyword.
- **Method Name** is the identifier of the method.
- **Parameter List** is a comma separated list of parameters. It contains the data type and variable name.
- **Method Body** contains all the actions to be performed.
- **Method Signature** includes the method name and parameter list.

Constructors

- Special member function -- same name as the class name
  - initialization of data members
  - Does NOT return a value
  - Implicitly invoked when objects are created
  - Can have arguments
  - Cannot be explicitly called
  - Multiple constructors are allowed
  - If no constructor is defined in a program, default constructor is called
  - no arguments

Constructors

- Syntax

```
class ClassName {
    ClassName() {
    }
}
```

- Example

```
Public class MyClass {
    int num;
    MyClass() {
        num = 100;
    }
}
```

Constructors

- Example (Default constructor)

```
Public class MyClass {
    int num;

    MyClass() {
        num = 100;
    }
}
```

```
public class TestMyClass {
    public static void main(String args[]) {
        MyClass test1 = new MyClass();
        MyClass test2 = new MyClass();
        System.out.println(test1.num + " " + test2.num);
    }
}
```

Constructors

- Example (Parameterized Constructor)

```
Public class MyClass {
    int num;

    MyClass(int i) {
        num = i;
    }
}
```

```
public class TestMyClass {
    public static void main(String args[]) {
        MyClass test1 = new MyClass(100);
        MyClass test2 = new MyClass(200);
        System.out.println(test1.num + " " + test2.num);
    }
}
```

Constructors

- Example (Constructor overloading)

```
Public class MyClass {
      int num;

      MyClass() {
            num = 100;
      }
      MyClass(int i) {
            num = i;
      }
}
```

```
public class TestMyClass {
      public static void main(String args[]) {
            MyClass test1 = new MyClass();
            MyClass test2 = new MyClass(200);
            System.out.println(test1.num + " " + test2.num);
      }
}
```

Comparison of OOPs with other programming styles

3 primary types of programming languages

- Unstructured Programming Languages
  - Earliest and primitive languages
  - Have sequential flow of control
  - Code is repeated
- Structured Programming Languages
  - No sequential flow of control
  - Use functions ; therefore, can reuse code
- Object Oriented Programming
  - Combines data & action together
  - Greater level of reuse

Comparison of OOPs with other programming styles

Scenario

- Initiate with
  - Account ID = 100
  - Account Balance = 100 000
- Deposit 10 000 to the account
- Display account number
- Display balance
- Withdraw 5 000
- Display account number
- Display balance

Comparison of OOPs with other programming styles

Unstructured Programming Language

```
int account_id = 100
int account_balance = 100 000
account_balance += 10 000
print("Account ID : ", account_id )
print("Account Balance", account_balance)
account_balance -= 5 000
print("Account ID : ", account_id )
print("Account Balance", account_balance)
```

Comparison of OOPs with other programming styles

Structured Programming Language

```
int account_id = 100
int account_balance = 100 000
account_balance += 10 000
showData()
account_balance -= 5 000
showData()
```

```
showData(){
  print("Account ID : ", account_id )
  print("Account Balance", account_balance)
}
```

Comparison of OOPs with other programming styles

OOP Language

```
class Account{
   int account_id
   int account_balance

   public void showData(){
      print("Account ID : ", account_id )
      print("Account Balance", account_balance)
   }
}
```

Advantages of OOP

- Easy to understand , develop and maintain
- Can hide data
- Provide ability to simulate real world events more effectively
- Objects created for OO programs can be reused.
- Offers clear modular structure

Example 1

Write a class called **circle**. It contains:

- Two **private** instance variables: **radius** (of the type **double**) and **color** (of the type **String**), with default value of **1.0** and "**red**", respectively.
- Write two **overloaded** constructors - a **default** constructor with no argument, and a constructor which takes a double argument for radius.
- Write two public methods: **getRadius()** and **getArea(),** which return the radius and area of this instance, respectively.

Example 1: More basic OOP concepts

Constructor: Modify the class **Circle** to include a third constructor for constructing a **Circle** instance with two arguments – a **double** for **radius** and **String** for **color.**

```
public Circle (double r, String c){……}
```

Getter: Add a getter for variable color for retrieving the color of this instance.

```
public String getColor(){……}
```

Example 1: More basic OOP concepts

public vs. private: In **TestCircle**, can you access the instance variable **radius** directly (e.g., System.out.println(c1.radius)); or assign a new value to radius (e.g., c1.radius=5.0)?

Try it out and explain the error messages.

Example 1: More basic OOP concepts

Setter: Is there a need to change the values of **radius** and **color** of a **Circle** instance after it is constructed? If so, add two **public** methods called *setters* for changing the **radius** and **color** of a **Circle** instance as follows:

```
// Setter for instance variable radius
public void setRadius(double newRadius) {
   radius = newRadius;
}

// Setter for instance variable color
public void setColor(String newColor) { ..... }
```

Example 1: More basic OOP concepts

"this" Keyword: Instead of using variable names such as **r** (for **radius**) and **c** (for **color**) in the methods' arguments, it is better to use variable names **radius** (for **radius**) and **color** (for **color**) and use the special keyword "**this**" to resolve the conflict between instance variables and methods' arguments.

```
// Instance variable
private double radius;

/** Constructs a Circle instance with the given radius and default color */
public Circle(double radius) {
   this.radius = radius;   // "this.radius" refers to the instance variable
                           // "radius" refers to the method's parameter
   color = "red";
}

/** Sets the radius to the given value */
public void setRadius(double radius) {
   this.radius = radius;   // "this.radius" refers to the instance variable
                           // "radius" refers to the method's argument
}
```

Example 1: More basic OOP concepts

Method toString():

well-designed Java class should contain a **public** method called **toString()** that returns a description of the instance (in the return type of **String**).

The **toString()** method can be called explicitly (via *instanceName*.toString()) just like any other method; or implicitly through **println()**.

If an instance is passed to the println(*anInstance*) method, the toString() method of that instance will be invoked implicitly.

Example 1: More basic OOP concepts

Method toString():

```
/** Return a self-descriptive string of this instance in the form of Circle[radius=?,color=?] */
public String toString() {
   return "Circle[radius=" + radius + " color=" + color + "]";
}
```

```
Circle c5 = new Circle(5.5);
System.out.println(c5.toString());   // explicit call
```

```
Circle c6 = new Circle(6.6);
System.out.println(c6.toString());  // explicit call
System.out.println(c6);             // println() calls toString() implicitly, same as above
System.out.println("Operator '+' invokes toString() too: " + c6);  // '+' invokes toString() too
```

Example 2

- Write a program to print the area and perimeter of a triangle having sides of 3, 4 and 5 units by creating a class named 'Triangle' without any parameter in its constructor.

Example 2

```
class Triangle{
private int length, width, height;

//Default Constructor
Triangle(){
Length=3; width=4; height=5;
}

//Parameterized Constructor
Triangle(int l, int w, int h){
Length=l; width=w; height=h;
}

int area(){
return length*width*height;
}

int perimeter(){
return length+width+height;
}
```

Example 2

```
//Setters
Void setlength(int l){
length=l;
}
Void setwidth(int w){
width=w;
}
Void setheight(int h){
height=h;
}
```

```
//Getters//
int getlength(){
return length;
}
int getwidth(){
return width;
}
int getheight(){
return height;
}
```

Example 2

```
Public class labexercise {
Public Static void main( String ags…) {
Triangle t=new Triangle();
Triangle t1=new Triangle(5,6,7);
System.out.println(“ Area of Triangle= “ +t.area());/// 60
System.out.println(“ Perimeter of Triangle= “ +t.perimeter());
System.out.println(“ Area of Triangle= “ +t1.area());
System.out.println(“ Perimeter of Triangle= “ +t1.perimeter());
t.setlength(12);
System.out.println(“ Area of Triangle= “ +t.area());///240
System.out.println(“ Area of Triangle= “ +t.area()+ “ And its width is
“+t.getwidth());
}
}
```

In-class Quiz (Quiz1)

- Write a class Rectangle. The class has attributes length and width, each of which defaults to 1. Provide methods that calculate the perimeter and the area of the rectangle. Provide set and get methods for both length and width. The set methods should verify that length and width are each floating-point numbers greater than or equal to 0.0 and less than 20.0. Write a program to test class Rectangle.

Activity 1 (Homework)

- Write a class called Employee that includes three pieces of information as instance variables - a first name, a last name, and a monthly salary. Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display the yearly salary for each Employee. Then give each Employee a 10% raise and display each Employee's yearly salary again

Activity 2 (Homework)

- Write a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables—a part number, a part description, a quantity of the item being purchased (type int), and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice's capabilities.

THANK YOU !