Exercise 2

Object-oriented programming

Part 1: Introduction to Java **27**

- Part 1.1: Installing JDK SE 27
- Part 1.2: Installing IntelliJ IDEA CE 27
- Part 1.3: Writing your first Java application 28
- Part 1.4: Java syntax 29
- Exercise: Writing simple Java applications 30

Part 2: Object-oriented design models **31**

- Exercise: Designing object-oriented models 31

Part 3: Modeling classes using UML notation **32**

- Exercise: Designing UML diagrams 32

Part 4: Defining classes and creating objects **33**

- Exercise: Defining classes using Java 34

Part 5: Types of classes and methods **36**

- Part 5.1: Concrete classes and methods 36
- Part 5.2: Abstract classes and methods 36
- Part 5.3: Final keyword 37
- Exercise: Review questions 37

Part 6: Implementing inheritance in Java **40**

- Exercise: Extending classes in Java 41

Part 7: Implementing polymorphism in Java **42**

- Exercise: Overriding methods in Java 42

Part 8: Implementing interfaces in Java **44**

- Exercise: Implementing interfaces in Java 45

Part 9: Exception handling in Java **47**

- Exercise: Handling run-time errors 47

Part 10: Practice questions **48**

References **48**

Estimated time

24:00 Hours

Overview

It is assumed that you have basic programming skills and, you are familiar with procedural/structured programming languages (i.e. `C/C++`). The exercise leverages on this familiarity to introduce you object-oriented programming concepts using `Java`.

Objectives

After completing this exercise, you should be able to:

- Install `Java JDK` on a PC.
- Install `IntelliJ IDEA` or `Netbeans` IDE on a PC.
- Write `Java` applications.
- Explain and illustrate object-oriented programming concepts using `Java`.

Prerequisites

A basic understanding of structured programming OR completed Exercise 1: *Introduction to programming*.

Requirements

This exercise requires a PC that has Internet access and, has a Web browser installed (i.e. `Chrome`).

Exercise instructions

In this exercise, you complete the following tasks:

- Write and compile `Java` code excerpts as answers to exercise questions.

Part 1: Introduction to `Java`

`Java` is a high-level, class-based, object-oriented programming language whose programs can run on all platforms (i.e. Windows, Linux, MacOS) that support Java without the need for recompilation. `Java` was initially designed by James Gosling and first released in 1995. The current official `Java` developer is Oracle Corporation.

Part 1.1: Installing `JDK` SE

Before you start write `Java` code, you need to install the `Java` Development Kit (`JDK`) in your operating system (OS). There are numerous ways for installing different versions of `JDK` on any OS. This course recommends the Oracle `JDK`.

Oracle `JDK` is software binary implementation (or software development kit - SDK) of the `Java` platform (i.e. standard edition, enterprise edition, micro edition) released by Oracle Corporaton for `Java` developers using different systems (i.e. Solaris, Linux, MacOS, Windows etc.). It includes a private `Java` virtual machine (JVM) and other resources for development of `Java` applications.

- Follow this link to install the Oracle `JDK` Standard Edition (SE): `https://www.oracle.com/java/technologies/javase-downloads.html`.

Part 1.2: Installing `IntelliJ IDEA` CE

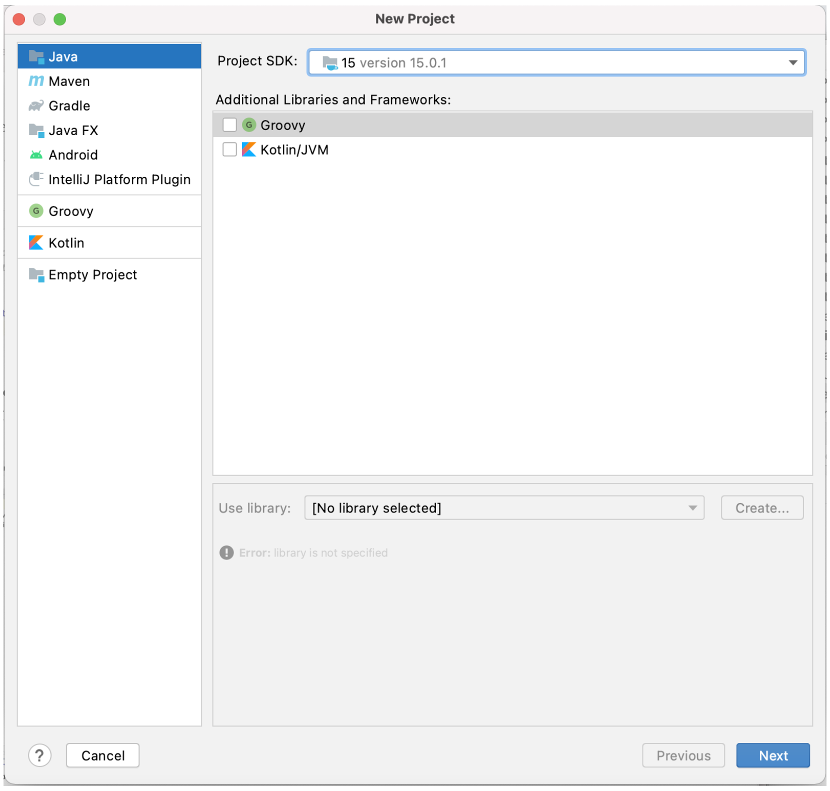
Again before getting started, you may need an IDE or a text editor that has been tailored to make editing, interpreting and executing `Java` code easy. This course recommends `IntelliJ IDEA` Community Edition (CE) IDE for these tasks.

- Installing `IntelliJ IDEA` Community Edition (CE) via Download, follow this link: `https://www.jetbrains.com/idea/download/`.

Part 1.3: Writing your first `Java` application

In this exercise, you write and run your first `Java` application. Follow the steps that follow:

1. Launch `IntelliJ IDEA` IDE, create a 'New Project' and edit the configuration details. Make sure you select your preferred `JDK` (already installed in Part 2.1).



Project SDK: 15 version 15.0.1

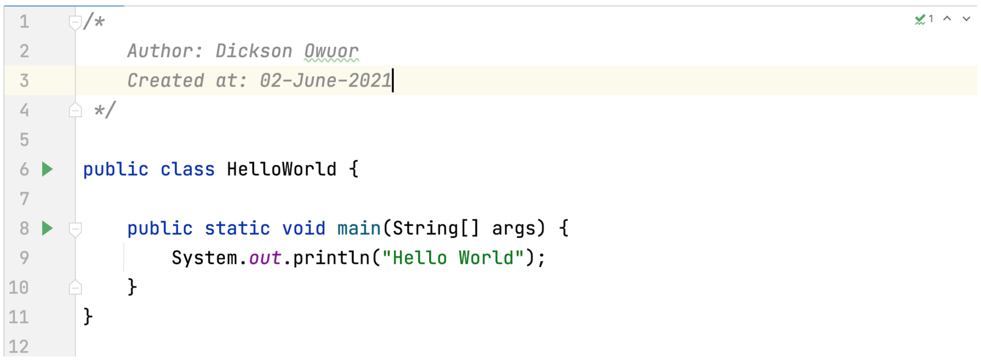
Additional Libraries and Frameworks:

- Groovy
- Kotlin/JVM

Use library: [No library selected]

Error: library is not specified

2. Create a new `Java` class from the 'File' menu (you may name it 'HelloWorld').
3. Write the code shown in the figure below.

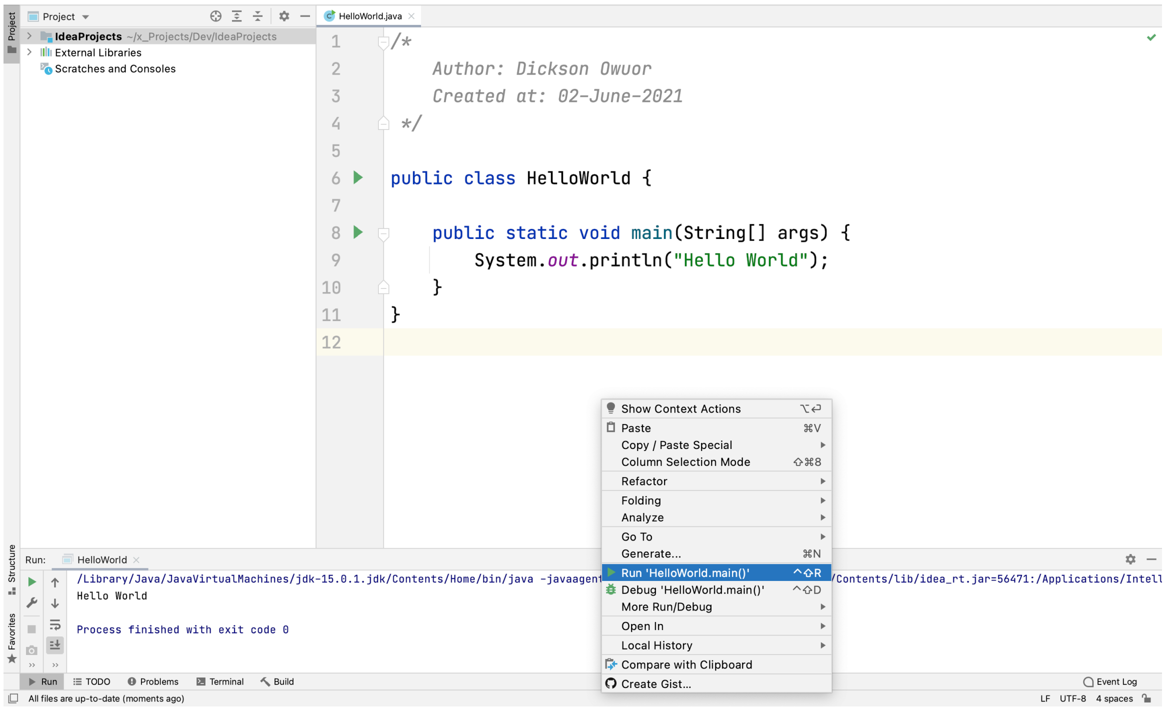


```
/*
    Author: Dickson Owuor
    Created at: 02-June-2021
 */

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

4. To compile and execute your code, right click on the editor and select **'Run HelloWorld.main()'** button as shown in the figure below. The results should be displayed in the Output window.



```java
/*
    Author: Dickson Owuor
    Created at: 02-June-2021
 */

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home/bin/java -javaagen    /Contents/lib/idea_rt.jar=56471:/Applications/Intell
Hello World

Process finished with exit code 0
```

Part 1.4: `Java` syntax

The syntax of `Java` is similar to `C` and `C++`. Below is the syntax to adhere to when writing `Java` code:

- `Java` is a case-sensitive language.
- Curly braces {} marks the beginning and the end of a block of code.
- Each code statement must end with a semicolon.

Exercise: Writing simple Java applications

Instructions

In this exercise, you complete the following tasks:

- Write and compile Java code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.
- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

Modify your Java code in Part 2.4 to implement the following tasks.

1. Write a simple java application that will display your name and gender on separate lines.
2. Write a simple java application that will prompt the user to enter three integer numbers. The program should calculate and display the sum, product and average of the numbers.
3. Write a simple java application that prompt a user to enter a number between 1 and 5. If the number is even, the program should display: **"Lucky Guess"**. If the number is odd, the program should display: **"Better Luck Next Time"**.
4. Write a simple java application to display all the odd numbers between 0 and 100. The program should also display the sum of the odd numbers.

Part 2: Object-oriented design models

Object-oriented design models software in terms similar to those people use to describe real-world objects. Object-oriented programming is the process of writing programs using a language that enables the implementation of object oriented design. `Java` is an example of an object-oriented language. The general object-oriented design concepts are:

- Classes & Objects
- Inheritance
- Polymorphism
- Data hiding
- Encapsulation
- Abstraction

Exercise: Designing object-oriented models

In this exercise, you model `Class-Object` relationship for real-world problems.

Instructions

In this exercise, you model `Class-Object` relationship for real-world problems. Complete the following tasks:

- Write all your answers on an A4 paper. Add your **Name**, **Student Number**, and **Date**.
- Scan your answers as a PDF and upload it the link provided in the e-learning.

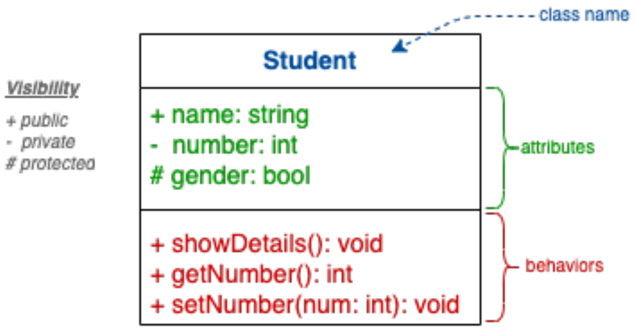
Questions

Attempt all the questions that follow:

1. Explain the difference between function overloading and function overriding.
2. Using the examples listed below model a `Class-Object` relationship using object-oriented design concepts. Illustrate and explain inheritance, polymorphism, encapsulation and abstraction.
   - student, undergraduate student, graduate student
   - Mary (is an undergraduate student), John (is a graduate student), Jane (is just a student)

Part 3: Modeling classes using `UML` notation

Unified Modeling Language (`UML`) is a general-purpose modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. Programmers can model classes using `UML` notation as shown in the figure below.



Exercise: Designing `UML` diagrams

Instructions

In this exercise, you model `Class-Object` relationship using `UML`. Complete the following tasks:

- Write all your answers on an A4 paper. Add your **Name**, **Student Number**, and **Date**.
- Scan your answers as a PDF and upload it the link provided in the e-learning.

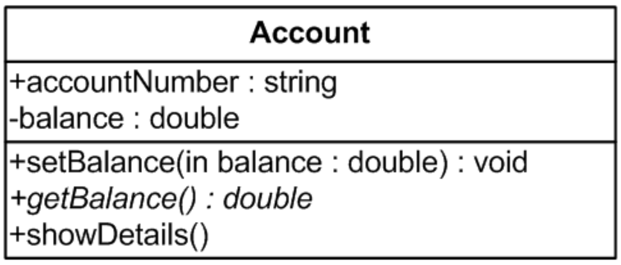
Questions

Attempt all the questions that follow:

1. Design a `UML` diagram(s) for the following entities: Employee, Fulltime, Parttime.
2. Illustrate inheritance, polymorphism and data hiding concepts using the `UML` diagram(s).

Part 4: Defining classes and creating objects

In this exercise, you write `Java` code that implement classes illustrated using `UML` diagrams.



Follow the steps below to implement the **Account** class shown in the `UML` diagram above:

1. Using your IDE, create class **Account.java** and write the code shown in the figure below.

```
// Account.java
package com.classes;

public class Account {

      private String account;
      private double balance;

      public Account(String account, double balance) {
      this.account = account;
      this.balance = balance;
   }

   public String getAccount() {
      return account;
   }

   public double getBalance() {
      return balance;
   }

   public void showDetails(){
      System.out.println("Account Number: " + account);
      System.out.println("Balance: " + balance);
   }
}
```

2. Within the same package, create class **Main.java** and write code that creates 2 objects of class **Account** as shown in the figure below.

3. Run **Main.main()** function to see the output.

```java
// Main.java
package com.classes;

public class Main {

   public static void main(String args[]){
      Account acc1 = new Account("ACC-00-001", 500);
      acc1.showDetails();

      Account acc2 = new Account("ACC-00-002", 1500);
      acc2.showDetails();
   }
}
```

Exercise: Defining classes using `Java`

Instructions

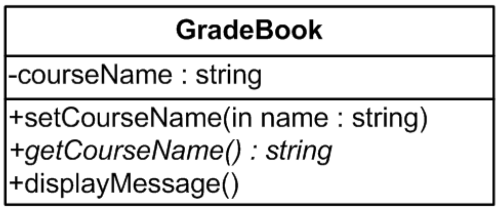
In this exercise, you complete the following tasks:

- Write and compile `Java` code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.
- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

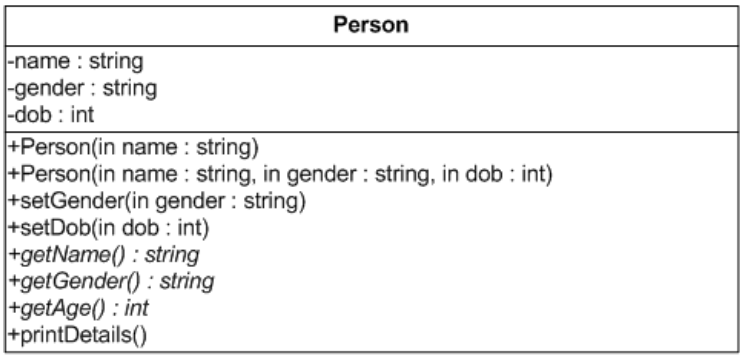
Attempt the questions that follow:

1. Write a simple `Java` application that will declare the class shown in the `UML` diagram below.
   - The method **displayMessage()** function should show a welcome message together with the course name (i.e "Welcome to Java Programming").
   - Create three objects and initialize their course names to: Calculus, C Programming and Business Ethics.
   - Display a welcome message for all the three courses.



2. Write a simple java application that will declare the class shown in the `UML` diagram below.

   - The **getAge()** method should calculate and return the present age of a person. Hint: Use dob (date of birth). The **printDetails()** method should display the full name of the person together with the gender and age in separate lines.
   - Create two objects from this class. Initialize the first one with the first constructor and, the second one with the second constructor. Use the **setGender()** and **setDob()** methods to initialize the gender and dob of the first object.
   - Display all the details of the two objects.



Part 5: Types of classes and methods

Part 5.1: Concrete classes and methods

A concrete class is a class that can instantiate objects.

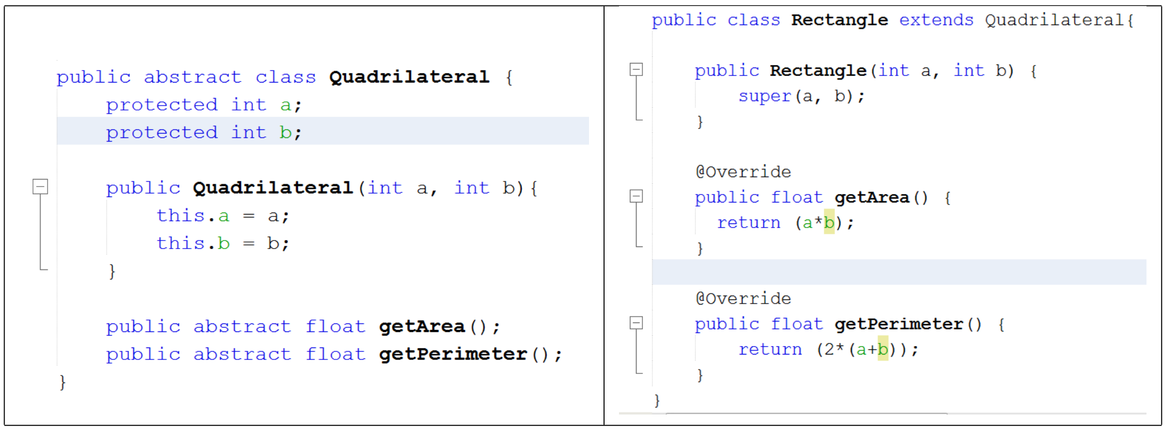
A concrete method is a method that has an implementation.

Part 5.2: Abstract classes and methods

So far most of the classes we have declared were concrete classes. In some cases, however, it is useful to declare classes for which the programmer never intends to instantiate objects. Such classes are called abstract classes. You make a class abstract by declaring it with keyword abstract. An abstract class contains one or more abstract methods.

Abstract classes are used as only super-classes in inheritance hierarchies. They cannot instantiate objects, because they are incomplete.

The purpose of abstract classes is to provide an appropriate superclass from which other classes can inherit and thus share a common design.



```java
public abstract class Quadrilateral {
    protected int a;
    protected int b;

    public Quadrilateral(int a, int b){
        this.a = a;
        this.b = b;
    }

    public abstract float getArea();
    public abstract float getPerimeter();
}
```

```java
public class Rectangle extends Quadrilateral{

    public Rectangle(int a, int b) {
        super(a, b);
    }

    @Override
    public float getArea() {
      return (a*b);
    }

    @Override
    public float getPerimeter() {
        return (2*(a+b));
    }
}
```

Abstract methods

An abstract method also uses the keyword abstract in its declaration. Abstract methods do not provide implementations.

Any class that contains any abstract method must be declared as an abstract class even if that class contains some concrete (non-abstract) methods. Abstract methods are meant to be overridden so they can process objects based on their types.

Part 5.3: Final keyword

In `Java`, variables declared as final cannot be modified after they are initialized (they become constants). It is also possible to declare methods, and classes with the final modifier.

A method that is declared final in a superclass cannot be overridden in a subclass. All subclasses use the same method implementation. Methods that are declared private are implicitly final, because it is impossible to override them (inaccessible). Methods declared static are also implicitly final.

A class that is declared final cannot be a superclass (it cannot be inherited)

Exercise: Review questions

Instructions

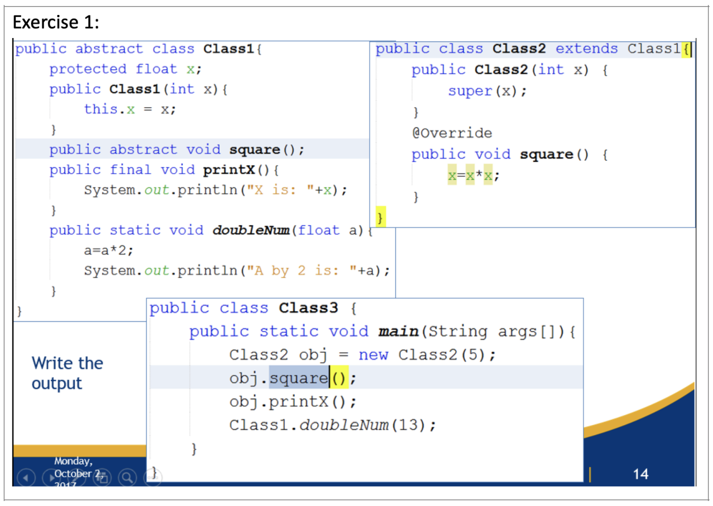
In this exercise, you complete the following tasks:

- Answer the questions in a Word document and upload it the link provided in the e-learning.
- Indicate your **Name** and **Student Number** in the Word document.

Questions

Attempt all the questions that follow.

1. Describe a static method? Give one circumstance where static methods can be used in a `Java` application.
2. List down the similarities and differences between keywords: this and super
3. Explain if there is any connection between abstract classes and polymorphism
4. Why is it not recommended to declare static methods as abstract.
5. Study the sample codes below, write down the outputs and explain how the output is arrived at.



Exercise 1:

```
public abstract class Class1{
    protected float x;
    public Class1(int x){
        this.x = x;
    }
    public abstract void square();
    public final void printX(){
        System.out.println("X is: "+x);
    }
    public static void doubleNum(float a){
        a=a*2;
        System.out.println("A by 2 is: "+a);
    }
}
```

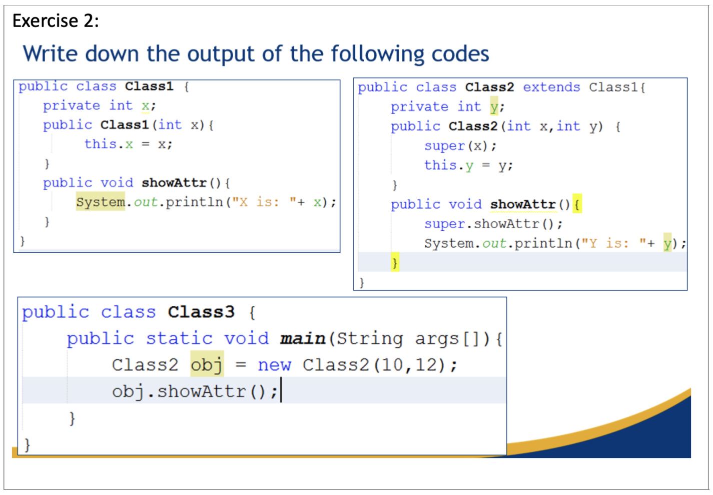
```
public class Class2 extends Class1{
    public Class2(int x) {
        super(x);
    }
    @Override
    public void square() {
        x=x*x;
    }
}
```

```
public class Class3 {
    public static void main(String args[]){
        Class2 obj = new Class2(5);
        obj.square();
        obj.printX();
        Class1.doubleNum(13);
    }
}
```

Write the output

Monday, October 2, 2017

14



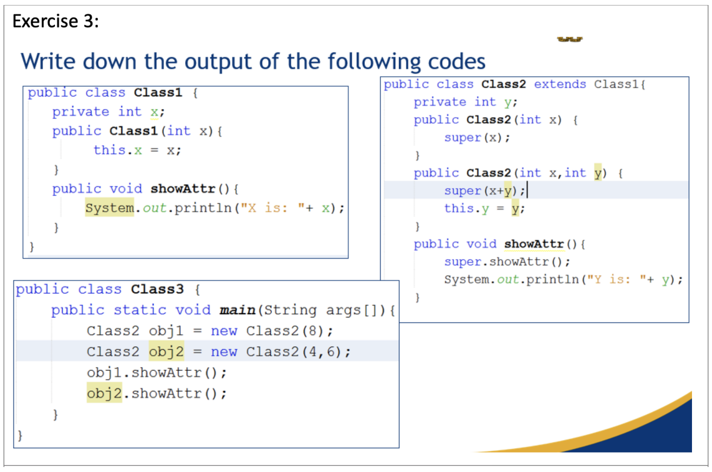
Exercise 2:

Write down the output of the following codes

```
public class Class1 {
    private int x;
    public Class1(int x){
        this.x = x;
    }
    public void showAttr(){
        System.out.println("X is: "+ x);
    }
}
```

```
public class Class2 extends Class1{
    private int y;
    public Class2(int x,int y) {
        super(x);
        this.y = y;
    }
    public void showAttr(){
        super.showAttr();
        System.out.println("Y is: "+ y);
    }
}
```

```
public class Class3 {
    public static void main(String args[]){
        Class2 obj = new Class2(10,12);
        obj.showAttr();
    }
}
```



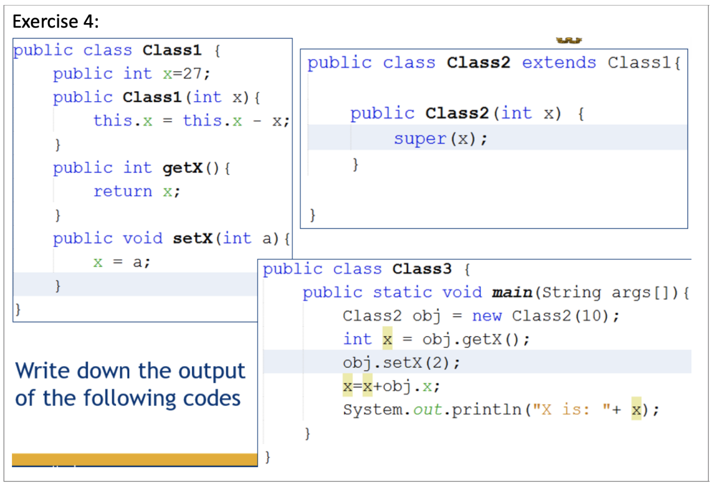
Exercise 3:

Write down the output of the following codes

```
public class Class1 {
    private int x;
    public Class1(int x){
        this.x = x;
    }
    public void showAttr(){
        System.out.println("X is: "+ x);
    }
}
```

```
public class Class2 extends Class1{
    private int y;
    public Class2(int x) {
        super(x);
    }
    public Class2(int x,int y) {
        super(x+y);
        this.y = y;
    }
    public void showAttr(){
        super.showAttr();
        System.out.println("Y is: "+ y);
    }
}
```

```
public class Class3 {
    public static void main(String args[]){
        Class2 obj1 = new Class2(8);
        Class2 obj2 = new Class2(4,6);
        obj1.showAttr();
        obj2.showAttr();
    }
}
```



Exercise 4:

```
public class Class1 {
    public int x=27;
    public Class1(int x){
        this.x = this.x - x;
    }
    public int getX(){
        return x;
    }
    public void setX(int a){
        x = a;
    }
}
```

```
public class Class2 extends Class1{

    public Class2(int x) {
        super(x);
    }

}
```

```
public class Class3 {
    public static void main(String args[]){
        Class2 obj = new Class2(10);
        int x = obj.getX();
        obj.setX(2);
        x=x+obj.x;
        System.out.println("X is: "+ x);
    }
}
```

Write down the output of the following codes

Part 6: Implementing inheritance in Java

Inheritance in Java is implemented using the **extends** keyword. The figures below illustrate how class **Motorcycle** is inherited by class **Scooter**.

```java
// Motorcycle.java
package com.inheritance;

public class Motorcycle {
   private String name;
   private float price;

   public Motorcycle(String name) {
      this.name = name;
   }

   public Motorcycle(String name, float price) {
      this.name = name;
      this.price = price;
   }

   public void setPrice(float price) {
      this.price = price;
   }

   public float getPrice() {
      return price;
   }

   public void showDetails(){
      System.out.println("Name: " + name);
      System.out.println("Price: " + price);
   }
}
```

```java
// Scooter.java
package com.inheritance;

public class Scooter extends Motorcycle{
   private int wheels;

   public Scooter(String name, float price, int wheels) {
      super(name, price);
      this.wheels = wheels;
   }

   public void showAttributes(){
      super.showDetails();
      System.out.println("Wheels: "+ wheels);
   }
}
```

```java
// Main.java
package com.inheritance;

public class Main {

   public static void main(String args[]){
      Motorcycle cycle1 = new Motorcycle("Honda", 6740);
      cycle1.showDetails();

      Scooter cycle2 = new Scooter("Suzuki", 2350, 3);
      cycle2.showDetails();
      cycle2.showAttributes();
   }
}
```

Exercise: Extending classes in `Java`

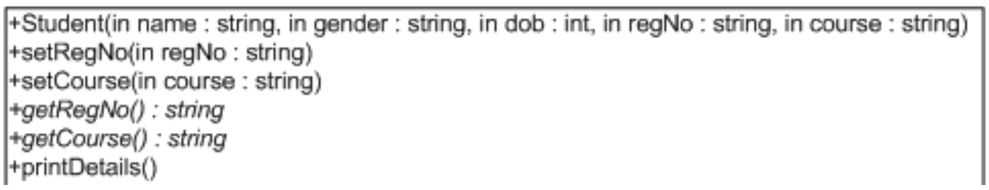
Instructions

In this exercise, you complete the following tasks:

- Write and compile `Java` code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.
- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

In this exercise, inherit Class **Person** in Part 3 and declare the class shown below. Write the `Java` code for class **Student**.



+Student(in name : string, in gender : string, in dob : int, in regNo : string, in course : string)
+setRegNo(in regNo : string)
+setCourse(in course : string)
+getRegNo() : string
+getCourse() : string
+printDetails()

1. Write a simple java application that will declare the class shown in the `UML` diagram below.
   - The **printDetails()** method should display the full name, gender, age, registration number and course of the student.
   - Create two objects from this class. Initialize them to any details of your choice.
   - Display all the details of the two objects.

Part 7: Implementing polymorphism in Java

Polymorphism in Java is implemented through inheritance and *Overriding* the inherited methods to perform different tasks. The figures below show how class **Scooter** overrides the **showDetails()** method inherited from class **Motorcycle**.

```java
// Scooter.java
package com.inheritance;

public class Scooter extends Motorcycle{
   private int wheels;

   public Scooter(String name, float price, int wheels) {
      super(name, price);
      this.wheels = wheels;
   }

   @Override
   public void showDetails(){
      System.out.println("This is another method");
      System.out.println("Wheels: "+ wheels);
   }
}
```

```java
// Main.java
package com.inheritance;

public class Main {

   public static void main(String args[]){
      Motorcycle cycle1 = new Motorcycle("Honda", 6740);
      cycle1.showDetails();

      Scooter cycle2 = new Scooter("Suzuki", 2350, 3);
      cycle2.showDetails();
   }
}
```

Exercise: Overriding methods in Java

Instructions

In this exercise, you complete the following tasks:

- Write and compile Java code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.

- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

Attempt all the questions that follow.

1. Using `Java`, define an abstract class **Person** with the following characteristics:
   - **attributes:** name, gender, date of birth.
   - **methods:** getAge():int - final, processLoan():double - abstract.
2. Using `Java`, define an abstract class **Student** which extends class **Person** and implement the method **processLoan()** according to the table shown below.
   - **attributes:** number of courses.

Age Bracket (years)	**Maximum Loan**
Below 10	1000
10 - 18	10, 000
18 - 24	100, 000

Part 8: Implementing interfaces in Java

Interfaces in Java is implemented using the keywords: **interface** and **implements**. The figures below illustrate how interface **Payable** is implemented in classes **Employee** and **Invoice**.

```
// Payable.java
package com.interfaces;

public interface Payable {
   double getPaymentAmount();
}
```

```
// Employee.java
package com.interfaces;

public class Employee implements Payable{
   private String fname;
   private String lname;
   private double salary;

   public Employee(String fname, String lname, double salary) {
      this.fname = fname;
      this.lname = lname;
      this.salary = salary;
   }

   public String getFname() {
      return fname;
   }

   @Override
   public double getPaymentAmount() {
      return salary;
   }
}
```

```
// Invoice.java
package com.interfaces;

public class Invoice implements Payable{
   private String itemName;
   private int quantity;
   private double pricePerItem;

   public Invoice(String itemName, int quantity, double pricePerItem) {
      this.itemName = itemName;
      this.quantity = quantity;
      this.pricePerItem = pricePerItem;
   }

   public String getItemName() {
      return itemName;
   }

   @Override
   public double getPaymentAmount() {
      return (quantity*pricePerItem);
   }
}

// Main.java
package com.interfaces;

public class Main {

   public static void main(String[] args){
      Invoice inv = new Invoice("Dress", 2, 1024.5);
      Employee emp = new Employee("John", "Doe", 5350);

      System.out.println(inv.getItemName() + "costs: " + inv.getPaymentAmount());
      System.out.println(emp.getFname() + "earns: " + emp.getPaymentAmount());
   }
}
```

Exercise: Implementing interfaces in Java

Instructions

In this exercise, you complete the following tasks:

- Write and compile Java code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.
- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

Attempt all the questions that follow.

1. Using Java, define interface **Span**:
   - **method:** number of years.
2. Using Java, define a concrete class **Course** with the following characteristics:
   - **attributes:** course name, course code, total hours.

- **methods:** getTotalHours():int.

3. Implement interface **Span** in class **Course** to calculate the duration of the course in years. [Hint – convert the total hours to years].
4. Implement the interface **Span** in class **Student** (in Part 6) to calculate the number of years a student will take to finish their studies. [Hint – assume that each course takes about 3 months to complete]
5. Create an object of class **Student** and **Course** and display their life spans.

Part 9: Exception handling in Java

Exceptions in Java are handled through the *try-catch-finally* structure. The figure below shows the syntax of this structure.

```
// Class3.java
package com.exception;

import java.util.InputMismatchException;
import java.util.Scanner;

public class Class3 {
   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int x;

      try {
         x = in.nextInt();
         System.out.println("You entered: "+x);
      }catch (InputMismatchException e){
         System.out.println("Please enter an integer.");
      }finally {
         System.out.println("Thank you!");
      }
   }
}
```

Exercise: Handling run-time errors

Instructions

In this exercise, you complete the following tasks:

- Write and compile Java code excerpts as answers to exercise questions.
- Add a comment (in each code excerpts) with the following information: **Name**, **Student Number**, and **Date**.
- Paste code excerpts in a Word document and upload it the link provided in the e-learning.

Questions

This exercise tests 2 concepts: (1) difference between **throw** and **throws** keywords and (2) exception inheritance. Attempt all the questions that follow.

1. Create a class with a **main()** that will throw an object of class **Exception** inside a try block. Give the constructor for Exception a String argument. Catch the exception inside

a catch clause and print the String argument. Add a finally clause and print a message to prove you were there.

2. Create your own exception class using the “extends” keyword (inherit class **Exception**). Write a constructor for this class that takes a String argument and stores it inside the object with a String reference. Write a method that prints out the stored String. Create a try-catch clause to exercise your new exception.
3. Write a class with a method that throws an exception of the type created in question 2. Try compiling it without an exception specification to see what the compiler says. Add the appropriate exception specification. Try out your class and its exception inside a try-catch clause.

Part 10: Practice questions

Attempt the questions that follow:

1. Write a program that displays all the *odd numbers* between 0 and 100. The program should also display the sum of the odd numbers.
2. Write a program that will calculate and display the *Factorial* of 10.
3. Write a program that will display the *Fibonacci sequence* of 10.

References

[1] https://en.wikipedia.org/wiki/Java_(programming_language)

[2] https://en.wikipedia.org/wiki/Java_Development_Kit

[3] https://www.w3schools.com/java/default.asp

[4] https://en.wikipedia.org/wiki/Unified_Modeling_Language

[5] http://www.linuxtopia.org/online_books/programming_books

End of exercise