MACQUARIE University

Department of Computing

ITEC625 Fundamentals of Computer Science
Workshop - Classes and Objects

Learning outcomes

By the end of this session, you will know some of Java basics. In particular, you will be able to design and write simple Java classes.

Questions

1. **Import-Export**

   It is important to know how to import Java projects from archive files (.jar/ .zip) and how to export your project(s) into archive files.

   For this exercise, download the file `classesAndObjectsProgram.zip` from iLearn but DO NOT unzip/open it.

   a. Click “File" –> “Import" –> “Existing Projects into Workspace" **and NOT "Archive file"**.
   b. Select option “Select Archive file" and click on “Browse"
   c. Choose the archive files (“.zip") that contains project(s) you want to open. Please note an archive file may contain multiple projects and click “ok"
   d. Check all projects you want to import
   e. Click “Finish"

   You should see a project `classesAndObjects` if correctly imported.

   Next, we’ll learn how to export a project.

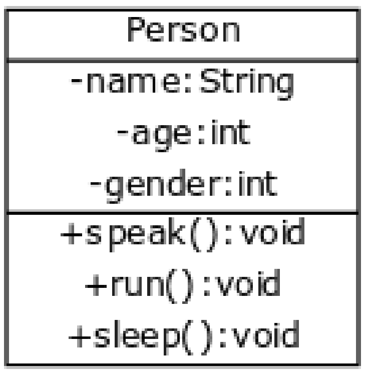
   a. Click “File" –> “Export" –> “General" –> “Archive file"
   b. Select all projects you want to export in the archive file in the left panel
   c. In the “To archive file" section, choose file path and name
   d. Click “Finish"

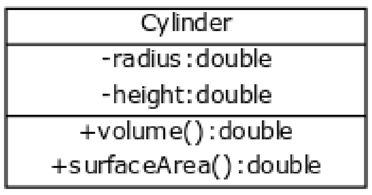
   Export the project `classesAndObjects` to an archive file `exported.zip`.

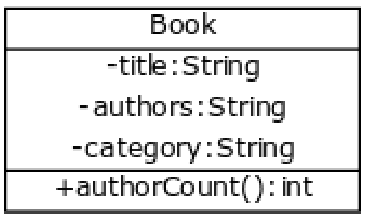
2. Design classes (no implementation) that encapsulate the following real life entities. Add up to three instance variables for each class. Select the three most important attributes if you think a class has more than three attributes. Describe your design in terms of a UML class diagram as shown in the lecture.

   a. `Person`
   b. `Cylinder`
   c. `Book`

Solution:







3. (a) Consider the following class definition,

```
public class Car {
        public String model;
        public int price;
}
```

In a client code (outside the class `Car`),

a. declare and instantiate an object `myCar` of class `Car`.
b. Assign the value `"Corolla"` to the instance variable `model` and the value `21999` to the instance variable `price` of object `myCar`.

Solution:

```
Car myCar = new Car();
myCar.model = ``Corolla'';
myCar.price = 21999;
```

(b) Consider the following class definition,

```java
public class Date {
        public int day, month, year;
}
```

In a client code (outside the class `Date`),

a. Declare and instantiate an object `graduation` of class `Date`.
b. Assign values to instance variables of object `graduation` such that it represents the date 13th April, 2011.

Solution:

```java
Date graduation = new Date();
graduation.day = 13;
graduation.month = 4;
graduation.year = 2011;
```

4. (a) Consider the following class definition,

```
public class Time {
        public int hour, minute, second;
}
```

Explain why it's a bad idea for the instance variables to be public, by writing a client that is malicious and assigns invalid values to the daa members of `Time` object.

Solution:

```
Time myTime = new Time();
time.hour = 888;
time.minute = -54;
time.second = -1729;
```

(b) Solve the problem of `public` instance variables in the previous part by first changing visibility of the instance variables of class `Time` to `private` and then adding getters and setters.

- The setter for `hour` should constrain the passed value in the range [0, 23]. That is,
  - if the passed value is less than 0, `hour` should become 0, otherwise,
  - if the passed value is more than 23, `hour` should become 23, otherwise,
  - `hour` should become the passed value.
- Similarly, the setters for `minute` and `second` should constrain the passed value in the range [0, 59].

Solution:

```
public class Time {
        private int hour, minute, second;

        //setters
        public void setHour(int hour)  {
                if(hour >= 0 && hour <=23)
                        this.hour = hour;
                else
                        this.hour = 0;
        }

        public void setMinute(int minute)  {
                if(minute >= 0 && minute <=59)
                        this.minute = minute;
                else
                        this.minute = 0;
        }

        public void setSecond(int second) {
                if(second >= 0 && second <=59)
                        this.second = second;
                else
```

```
                                this.second = 0;
          }

          //getters
          public int getHour() {
                  return hour;
          }

          public int getMinute() {
                  return minute;
          }

          public int getSecond() {
                  return second;
          }
}
```

(c) Declare and instantiate an object `myTime` of class `Time` written in the previous part. Assign values to the instance variables such that it represents the time 19:30:45 (half past seven in the evening and another 45 seconds).

Solution:

```
Time myTime = new Time();
myTime.setHour(19);
myTime.setMinute(30);
myTime.setSecond(45);
```

(d) Declare and instantiate an object `yourTime` of class `Time` written in the previous part. Assign 95 to `hour`, -78 to minute, and 55 to second. Display all instance variables on the console. What time would `yourTime` represent?

Solution:

```
Time yourTime = new Time();
yourTime.setHour(95);
yourTime.setMinute(-78);
yourTime.setSecond(55);
System.out.println(yourTime.getHour()); //23
System.out.println(yourTime.getMinute()); //0
System.out.println(yourTime.getSecond()); //55
```

(e) List the mistakes (syntactical and logical) in the following constructor for class `Time` -

```
public void time(int h) {
        hour = h;
        minute = 0;
        second = 0;
}
```

Solution:

- Constructor has no return type
- Name of constructor should be **exactly** the same as the class name.
- Constructor should use setters to assign values to instance variables.

Fixed constructor,

```
public Time(int h) {
        setHour(0);
        setMinute(0);
        setSecond(0);
}
```

(f) Add two constructors to class `Time` with the following requirements:

- A constructor that is passed three parameters, one for each instance variable.
- A constructor that is passed two parameters, for `hour` and `minute`, and sets `second` to 0.

Solution:

```
public Time(int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(s);
}

public Time(int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(0); //default value
}
```

(g) Assuming the two constructors have been added to class `Time` according to previous part. Will the following program run successfully, or result in a compilation error? Explain your answer. Also, if there is a compilation error, what should be done to fix it?

```
Time ourTime = new Time();
```

Solution: It will result in a compilation error, since once parameterized constructors are defined, Java expects us to define the default constructor as well, and the default constructor that Java provides is no longer valid. The solution, therefore, is to add a default constructor.

```
public Time() {
        setHour(0);
        setMinute(0);
        setSecond(0);
}
```

5. **Programming Exercise**

   a. Write a class definition for a `Cylinder`, as represented by its radius and height. It should include,

      (a) Correct class header.

      (b) instance variables with appropriate visibility and data types.

      (c) Getters

      (d) Setters, with appropriate validation.

      (e) Constructors

         i. With no parameters. Set each instance variable to 1.

         ii. With two parameters, one for each instance variable.

      (f) A method `volume()` that returns the volume of the cylinder. The formula for volume of a cylinder with radius r and height h is $\pi \times r^2 \times h$ (where r is the radius of the circle). The value of π in Java is given by `Math.PI`, and square of a number n is calculated as `n*n` or `Math.pow(n,2)`, and NOT **n^2** or **Math.sqr(n)** .

   b. Create a Client `CylinderClient` that performs the following tasks -

      (a) Create a `Cylinder` object `c1` that has a radius of 2.5 and height of 4.7

      (b) Display the volume of `c1`.

      (c) Increase the radius of `c1` by 2.1

      (d) Display the volume of `c1`.

Solution:

Cylinder.java

```
public class Cylinder {
        private double radius, height;

        public void setRadius(double r) {
                radius = Math.abs(r);
        }

        public void setHeight(double h) {
                height = Math.abs(h);
        }

        public double getRadius() {
                return radius;
        }

        public double getHeight() {
                return height;
        }

        public Cylinder() {
                setRadius(1);
                setHeight(1);
        }

        public Cylinder(double r, double h) {
                setRadius(r);
                setHeight(h);
        }

        public double volume() {
                return Math.PI * radius * radius * height;
        }
}
```

CylinderClient.java

```
public class CylinderClient {
        public static void main(String[] args) {
                Cylinder c1 = new Cylinder(2.5, 4.7);
                System.out.println(c1.area());
                c1.setRadius(c1.getRadius() + 2);
                System.out.println(c1.area());
        }
}
```

6. **The `this` keyword**
   Consider the following class definition,

```java
public class Circle {
        private double radius;

        //assume getters, setters,
        //assume default and parameterized constructors

        public int compareTo(Circle other) {
                if(this.radius > other.radius)
                        return 1;
                if(this.radius < other.radius)
                        return -1;
                return 0;
        }
}
```

Further, consider the following client,

```java
public class Client {
        public static void main(String[] args) {
                Circle c1 = new Circle(4.5);
                Circle c2 = new Circle(4.8);
                int status = c2.compareTo(c1);
}
```

Explain which is the `this` object and which is the `obj` object in the context of the method call `c2.compareTo(c1)` by drawing a memory diagram.

Solution: `this` is a shallow copy of the calling object `c2` while `other` is a shallow copy of parameter object `c1`

7. **The `compareTo` method**
   Complete the `compareTo` method in class `Box` such that it returns,

   - 1, if the calling object has a higher capacity than the parameter object
   - -1, if the calling object has a lower capacity than the parameter object
   - 0, if the calling object has the same capacity as the parameter object

Solution:

```java
public int compareTo(Box other) {
        if(capacity > other.capacity)
                return 1;
        if(capacity < other.capacity)
                return -1;
        return 0;
}
```