MACQUARIE University

Department of Computing

COMP125 Fundamentals of Computer Science
Workshop - Workshop - Classes and Objects: 2

Learning outcomes

Following are this week's learning outcomes,

a. Design classes

b. Implement classes (write class definition), including,

   (a) instance variables

   (b) getters and setters

   (c) constructors, that call setters

   (d) `toString()`, `compareTo()`, `equals()` methods

c. Understanding the `this` keyword

Questions

1. a. Write a class definition for a `Cube`. A cube is a 3-dimensional object, enclosed by 6 equal squares. A cube is characterised by the length of each side (all sides have equal length).

   (a) Correct class header.

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

   (c) Getters

   (d) Setters

   (e) Constructors

      i. With no parameters. Length of each side is set to 1.

      ii. With one parameter for length of each side.

   (f) A method `volume()` that returns the volume of the cube given by $side * side * side$.

   (g) The method `toString()` that returns the `String` representation of object of class `Cube`. For example, for a cube of sides 2.5, the method should return "`cube of size 2.5`".

   (h) The method `compareTo` that when passed another `Cube` object, returns 1 if the calling object has a larger side than that of the parameter object; returns -1 if the calling object has a smaller side than that of the parameter object; returns 0 if the calling object and the parameter object have the same sides.

Solution:

```
1 public class Cube {
2         private double side;
3
4         public double getSide() {
5                 return side;
```

```
        }

        public void setSide(double side) {
                this.side = Math.max(0, side);
        }

        public Cube() {
                setSide(1);
        }

        public Cube(double side) {
                setSide(side);
        }

        public double volume() {
                return side*side*side;
        }

        public String toString() {
                return "cube of size "+side;
        }

        public int compareTo(Cube other) {
                if(this.side > other.side)
                        return 1;
                if(this.size < other.side)
                        return -1;
                return 0;
        }
}
```

b. Write a piece of code outside the class `Cube` (in another class containing `public static void main(String[] args)` that does the following,

(a) Declare and instantiate an object `myCube` of class `Cube` with sides 2.5

(b) Store the volume of `myCube` in a variable `myVolume`.

(c) Declare and instantiate an object `yourCube` of class `Cube` with sides 4.5

(d) Store the volume of `yourCube` in a variable `yourVolume`.

(e) Store the comparison of `myCube` with `yourCube` in a variable `myCubeComparedToYours`. The value of`myCubeComparedToYours` should be -1 if defined and called correctly.

Solution:

```
public class Client {
        public static void main(String[] args) {
                Cube myCube = new Cube(2.5);
                int myVolume = myCube.volume();
                Cube yourCube = new Cube(4.5);
                int yourVolume = myCube.volume();
                int myCubeComparedToYours = myCube.compareTo(yourCube);
        }
}
```

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

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

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

        public boolean equals(Object obj) {
                if(obj instanceof Circle) {
                        Circle other = (Circle)obj;
                        if(radius == other.radius) {
                                return true;
                        }
                }
                //in all other cases
                return false;
        }
}
```

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);
                boolean status = c2.equals(c1);
}
```

Explain which is the `this` object and which is the `obj` object in the context of the method call `c2.equals(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`

3. **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;
}
```

4. **The `equals` method**
   Complete the `equals` method in class `Box` such that it returns,
   - `false` if the parameter object is not a Box object
   - otherwise,

- `true`, if the calling object has the same capacity as the parameter object
- `false`, if the calling object has a different capacity from the parameter object

Solution: Assuming `compareTo` has been implemented,

```
public boolean equals(Object obj) {
        if(obj instanceof Box) {
                Box other = (Box)obj;
                if(compareTo(other) == 0) {
                        return true;
                }
        }
        //in all other cases
        return false;
}
```

Assuming `compareTo` has NOT been implemented,

```
        public boolean equals(Box other) {
                if(obj instanceof Box) {
                        Box other = (Box)obj;
                        if(radius == other.radius) {
                                return true;
                        }
                }
                //in all other cases
                return false;
        }
```

5. **JUnit test**

   a. Run the JUnit test `testVolume` in JUnit test class `SphereTest`. Correct the method `volume` in class `Sphere` so that the test passes. The description for the method's requirements is provided as method comment.

Solution:

```
        public double volume() {
                return 4 / 3.0 * Math.PI * radius * radius * radius;
        }
```

6. Assuming the existence of the following Line class, answer the questions after it.

```
        public class Line {
                private double x1, y1, x2, y2;
                //assume getters, setters, constructors

                public double getLength() {
                        return Math.sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2));
                }
        }
```

   (a) Write a JUnit test case to ensure the correctness of `getLength()`. You will need the assertion

   `assertEquals(expectedValue, valueReturnedByMethodUnderTest, tolerance)`

```
                Test
                public void testGetLength() {
```

```
                //to be completed
        }
```

(b) Define instance method `equals` that when passed an object of class `Line`, returns true if the length of the calling object and parameter object are the same, and false otherwise.

(c) Define instance method `compareTo` that when passed an object of class `Line`, returns 1 if the length of the calling object is more than that of parameter object, -1 if the length of the calling object is less than that of parameter object, and 0 if the length of the calling object and parameter object are the same.

Solution:

```
        public boolean equals(Object obj) {
                if(obj instanceof Line) {
                        if(getLength() == other.getLength())
                                return true;
                }
                //in all other cases
                return false;
        }

        public int compareTo(Line other) {
                if(getLength() > other.getLength())
                        return 1;
                if(getLength() < other.getLength())
                        return -1;
                return 0;
        }
```

Supplementary exercises (take-home exercises)

1. **Email address**

Complete the definition of class `EmailAddress` based on the requirements provided as comments in the class, and run the the client `EmailAddressClient`.

Solution:

```
package comp125;

/**
 * email address is stored as a combination of username and domain
 * for example, username = "gaurav.gupta"
 * domain = "mq.edu.au"
 * thus the username is "gaurav.gupta@mq.edu.au"
 * @author gauravgupta
 *
 */
public class EmailAddress {
        private String username, domain;

        public String getUsername() {
                return username;
        }

```

```
18          public String getDomain() {
19                  return domain;
20          }
21
22          public void setUsername(String username) {
23                  this.username = username;
24          }
25
26          public void setDomain(String domain) {
27                  this.domain = domain;
28          }
29
30          public EmailAddress(String username, String domain) {
31                  setUsername(username);
32                  setDomain(domain);
33          }
34
35          public EmailAddress() {
36                  setUsername("tba");
37                  setDomain("tba");
38          }
39
40          public String toString() {
41                  return username+"@"+domain;
42          }
43
44          public boolean equals(Object obj) {
45                  if(obj instanceof EmailAddress) {
46                          EmailAddress other = (EmailAddress)obj;
47                          if( username.equals(other.username) {
48                                  if(domain.equals(other.domain)) {
49                                          return true;
50                                  }
51                          }
52                  }
53                  //in all other cases
54                  return false;
55          }
56  }
```

2. **Bank account**

Define a class representing a bank account with methods to check the balance, to deposit and withdraw funds and to test whether the account is overdrawn.

In package `comp125`, there is an outline implementation of the class along with a set of tests for the different methods. Use this as the basis for your implementation. Read also the in-code comments for the details of what each method is expected to do.

The first thing you need to do is decide how the `BankAccount` class is going to store its data – the bank balance. You need to define an instance variable of the appropriate type in the class. (Label your instance variable `private`.)

Then you need to write the body of each of the methods. At each point, run the tests to see if what you've written works. Try to work on the tests from top to bottom, getting one working after another. You can use the debugger to walk through your program if it's not doing what you expect – remember to place a breakpoint in your class to stop execution when running under the debugger.

When you've passed all the tests, complete the `public static void main(String[] args)` method in `BankAccountClient` to make use of the bank account class. Make a bank account, add $42.5 to the account, then

withdraw $80.6 from it, display if it's overdrawn (`true`) or not (`false`) and display the balance. You can include such a `main` method inside your `BankAccount` class.

Please note that `BankAccountTest` uses an `assertEquals` assertion that has three parameters. The syntax is;

```
assertEquals(expectedDouble, returnedDouble, tolerance);
```

The assertion is successful (passes) if the `expectedDouble` and `returnedDouble` differ by at most `tolerance`. That is, $Math.abs(expectedDouble - returnedDouble) \leq tolerance$.

Solution:

```
public class BankAccount {
        public BankAccount() {
                setBalance(0);
        }

        /**
         * @return the balance of the account
         */
        public double getBalance() {
                return balance;  // You might want to change this statement.
        }

        /**
         * since balance can be negative or postive (or zero), there is really
         * no validation rule here
         * @param bal
         */
        public void setBalance(double bal) {
                balance = bal;
        }

        /**
         * Increase the balance by the amount specified.
         * @param amount
         */
        public void deposit(double amount) {
                setBalance(balance+amount);
        }

        /**
         * Decrease the balance by the amount specified. It's OK if the resulting
         * balance is negative (overdrawn).
         * @param amount
         */
        public void withdraw(double amount) {
                setBalance(balance-amount);
        }

        /**
         * @return true if the balance is negative, false otherwise
         */
        public boolean overdrawn() {
                return balance < 0;  // You might want to change this statement.
        }
}
```