University of Puerto Rico
Department of Electrical and Computer Engineering

ICOM 4015 Laboratory: Advanced Programming

Laboratory: Interfaces

Completed by:
ID:
Date:

Introduction

In this laboratory we will practice interfaces.

Create an Eclipse project with the provided Java files and run it.
The program is a small game. Experiment with it.
After you've seen the game, browse through the code.
Note: You should maximize the console view so that you see all the text.

Part 1.
Create a new interface, call it Readable. Make it so that it contains the following code:

```
public interface Readable
{
    void read();
}
```

Now, let's make the Sign class Readable. Add Readable to the implements clause of the Sign class, and add the "read" method and make it public. Your code should look like this:

```
public class Sign implements GameThing, Readable
{
    public String getName()
    {
        return "A wooden sign";
    }
    public String getDescription()
    {
        return "This is a small wooden sign. There seems to be some partially faded writing in it.";
    }
    public void read()
    {
        System.out.println("You can see the following message: \"John was here!\"");
    }
}
```

Make sure you notice what has changed when compared to the initial version of the Sign class.
You can change the message within read(), if you want.

Now let's update the main menu so that we can read things. Add the following to the code that displays the main menu:

```
    System.out.println("5. Read something");
```

Add the following case to the switch:

```
    case 5:
        System.out.println("Which thing do you want to read?");
        theRoom.listContents();
        System.out.print("Your choice? (Enter an unlisted number to go back to the main menu) >>>");
        thingChoice = keyboard.nextInt();
        System.out.println();
        if (thingChoice >= 1 && thingChoice <= theRoom.getThingCount())
```

```
        {
            GameThing thing = theRoom.getThingByIndex(thingChoice);
            if (thing instanceof Readable)
            {
                Readable readableThing = (Readable)thing;
                readableThing.read();
            }
            else
            {
                System.out.println("That's not something you can read!");
            }
            System.out.println();
        }
        break;
```

The condition "thing instanceof Readable" checks to see if the chosen object (a game thing) implements the Readable interface. Then, inside the "if", we need to treat the thing as a Readable. To do so, we use the following expression:

(Readable)thing

That expression refers to the same as that which "thing" refers to, but we're interpreting it as a Readable. That expression is known as a "cast expression". We know we can do this because the condition "thing instanceof Readable" was satisfied. Now, the statement

Readable readableThing = (Readable)thing;

makes the variable "readableThing", which is of type Readable, refer to the same as that to which "thing" refers to. Please note that the following is not allowed:

Readable readableThing = thing;

That's because the compiler knows that "thing" is a GameThing, but it does not know that "thing" is a Readable, that's why we need to tell it with the cast.
After readableThing has been made to point to "thing", we can call read() on it because it is a Readable. If we do this on the sign, a message will get displayed.

Run the game and test it by trying to read the box and the sign.

OK, let's create a Book class. It should not have a main() method.

```
public class Book
{

}
```

It should implement GameThing so that we can put it into the room. (If you look at the GameRoom class, the addThing method takes an argument of type GameThing.)

```
public class Book implements GameThing
{

}
```

You will see that Eclipse gives you an error. If you look at the code for the GameThing interface, you'd see that it looks like this:

```
public interface GameThing
```

```
{
    String getName();
    String getDescription();
}
```

Therefore, we should add those methods to our Book class, as public:

```
public class Book implements GameThing
{
    public String getName()
    {
        return "A dusty old book";
    }
    public String getDescription()
    {
        return "This book appears to be old and dusty. It's also quite heavy.";
    }
}
```

We've created a simple Book class! But we need to create an instance of it. Add the following towards the top of the AdventureEpsilon class (outside the method!):

```
    public static Book theBook = new Book();
```

We also need to add it to the room, or we won't see it. Add the following towards the top of the AdventureEpsilon class (inside the method!):

```
    theRoom.addThing(theBook);
```

Now run the program. You should be able to examine the book, but we can't read it yet because it hasn't been made Readable.

Part 2.

Exercise #1.

Make the Book class Readable. You need to update the implements clause, and add the only method that's part of the Readable interface. Which method is it? Look at the Readable interface to find out. How to implement the method inside the Book class? You can do it similar to that method in the Sign class. The message that gets displayed should be different. For example, you can say that it is too difficult to read, or that it has many color illustrations, or that it seems to be an unreadable magic book, or any other message that you prefer. Run the game and try reading the book. When you are done, copy and paste the whole Book class below.

//

(Your answer here)

//

Exercise #2.1.

Some things can be opened and closed. Create an Openable interface that has the following two method declarations:

```
    void open();
    void close();
```

Now make it so that the Book implements the Openable interface. You need to update the

implements clause and implement the open and close methods. Inside the open and close methods, use System.out.println to give a description of what happens when you open and close the book, respectively. For example, when you close the book it could say:

"As the book closes, it makes a loud 'thud' sound."

Please note, however, that if the book is already closed it should say:

"The book is already closed!"

In order to be able to accomplish that, you'll need to use the following boolean variable:

```
public boolean isOpen = false;     //Start out closed
```

Put the boolean variable inside the class, but outside the methods.

A similar pair of messages could be used when you try to open the book:

"The book opens with a creaky sound."

"The book is already open!"

You can change those messages to some other text that you prefer. Remember to update the isOpen variable as necessary! When you are done, copy and paste the whole Book class below.

//

(Your answer here)

//

Exercise #2.2.

Update the main menu to give two additional options:

6. Open something
7. Close something

Then create new cases 6 and 7 within the "switch" block that is located within the main method of class AdventureEpsilon: copy and paste the code from case 5, and adjust it so that the opening and closing activities work properly. You should also modify the text strings, such as the question "Which thing do you want to read?" to use the appropriate verb (open or close). When you run the game, the following should work properly:

* If you try to open the book when it is closed, it should work and display an appropriate message
* If you try to open the book when it is already open, it should just display an appropriate message
* If you try to close the book when it is open, it should work and display an appropriate message
* If you try to close the book when it is already closed, it should just display an appropriate message

When you are done, copy and paste the whole AdventureEpsilon class below.

//

(Your answer here)

//

Exercise #3.

Update the Book class so that getDescription() also tells you whether the book is currently open or closed. Then test the program by running it and using the option to *examine* things.

Also, update the Book class so that read() works as usual if the book is open, but, if the book is closed then it should say something along the lines of:

"I can't read closed books!"

Test the game by using the read option while it is open or closed.

When you are done, copy and paste the whole Book class below.

//

(Your answer here)

//

Exercise #4.

In ways similar to how the Book class was created and used:

* Create an Apple class (use any relevant name and description that you want)
* Create an instance of the Apple class (call it theApple). In which class do you do this?
* Make sure that the apple is in the room
* Test the game by examining the apple
When you are done, copy and paste the whole Apple class below, and not any other class.
//
(Your answer here)
//

Exercise #5.1.
Create an interface called Edible. It should have only the following method:

```
void eat();
```

Now, make the apple edible.
* Update the implements clause of the Apple class.
* Add the eat method (it should be public). Inside it, give a message about how the apple tasted, and also you should call the following:

```
AdventureEpsilon.theRoom.removeThing(AdventureEpsilon.theApple);
```

so that it does not show up anymore since it has been eaten.
When you are done, copy and paste the whole Apple class below, and not any other class.
//
(Your answer here)
//

Exercise #5.2.
Update the main menu to give the option to eat. Case 8 should ask "Which thing do you want to eat?". Run the game and check that the apple disappears after it is eaten.
When you are done, copy and paste the whole AdventureEpsilon class below.
//
(Your answer here)
//

Exercise #6.
Let's make it so that the sign is one made of chocolate instead of it being wooden. Change its getName() and getDescription() appropriately. Now, make it so that the sign is edible. It should disappear after being eaten. Also, how does it taste? When you are done, copy and paste the whole Sign class below.
//
(Your answer here)
//