Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

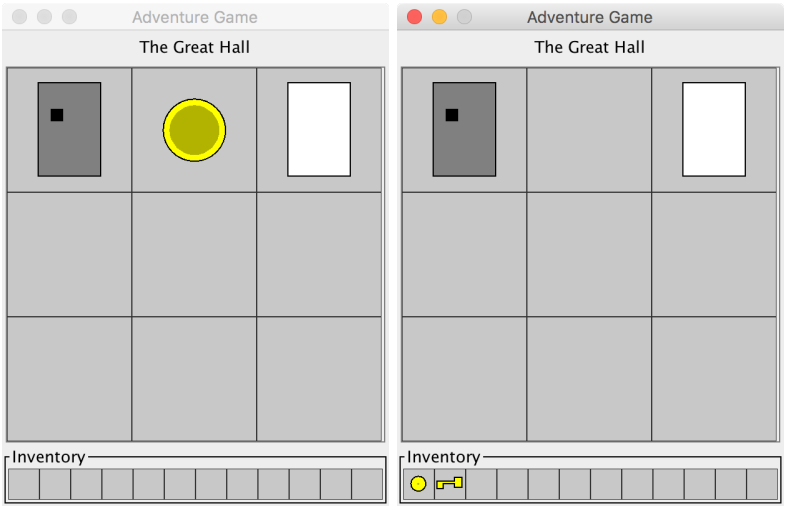
Lab Handout

Outline

The purpose of this lab is to get yet more experience with inheritance and polymorphism. In particular, you will use polymorphism to further develop a simple graphical adventure game. Before the end of the lab, you should submit your solutions via the *online submission system* which will automatically mark it. You may submit as many times as you like in order to improve your mark and the final deadline will be Friday @ 23:59.

An Adventure Game

In this lab you will be working with a very simple graphical adventure game. The adventure game world consists of Rooms connected together by Doors, and Items which are located in Rooms. The following illustrates two different views of the game:



Here, we can see two views of the game. In both cases, the player is in the "Great Hall". In the second case, we can see that he/she has picked up a few items, including a "Gold Coin" and a "Key".

The adventure game uses a simple text format to describe the game world, including the layout of rooms and the items they contain. This is implemented in the `GameFile` class, and an example is the following:

```
Room { name: "Dining Room" }
Room { name: "Lounge" }
Door { from: 0, to: 1 }
Coin { location: 1 }
```

This describes a game world with two rooms (the "Dining Room" and the "Lounge") with one door connecting them. A gold coin can be found in the "Lounge". Each line of a game file corresponds to a `GameFile.Item` organised in the following manner:

```
Kind { field: value, ... }
```

Here, the Kind of object is `"Room"`, `"Door"`, `"Coin"`, `"Key"`, etc. Finally, field is the name of a field for the object in question, whilst value gives a value for that field (either a string or integer). There can be more than on field for any given object.

Getting Started (30 minutes)

To get started, download the `adventure.jar` file from the course website and import this into an Eclipse project. The adventure game is provided with a Graphical User Interface (GUI), which you can run by right-clicking on `AdventureGame` and selecting "Run As⟶Java Application". You should find a game window similar to those above appears, and you can click on items in the game and perform actions.

Activity 1: Books (30 minutes)

The aim of this activity is to extend the adventure game with a new `Book` item. Each `Book` should have a title, and be described like this in the `GameFile` format:

```
Room { description: "Dining Room" }
Book { location: 0, title: "Great Expectations" }
```

This describes a room called the "Dining Room" which contains a book entitled "Great Expectations". The player should be able to `Pickup` and `Drop` a `Book`. The player should be able to `Read` a book when it is held in his/her inventory. Every book also has a description which is formed from the book's title. Before a book is read, the description for the above book is simply:

```
A book entitled "Great Expectations"
```

After a book has been read, its description should be updated as the following illustrates:

```
A book entitled "Great Expectations"; it looks like it has been read
```

What to do. You should create a new class called `Book` which extends `PickupableItem`. You should also make a small modification to the `AdventureGame` class in order to create instances of your `Book` class from the `GameFile.Items` which describe them. Having done this, you should find that more of the tests in `AdventureGameTests` now pass.

Activity 2: Lockable Doors (30 minutes)

The aim of this activity is to extend the adventure game with a new `LockedDoor` class which extends the `Door` class. Each `LockedDoor` should have a secret "code" so that keys with the corresponding code can can be used to unlock it. `LockedDoor`s are described as follows in the `GameFile` format:

```
Room { description: "Dining Room" }
Room { description: "Hall" }
LockedDoor { from: 0, to: 1, code: 123 }
```

This describes a locked door connected the "Dining Room" with the "Hall". The player should be able to `Unlock` and `Lock` the door, but only if he/she is carrying a key with the secret code "123".

What to do. You should create a new class called `LockedDoor` which extends `Door`. You should also make a small modification to the `AdventureGame` class in order to create instances of your `LockedDoor` class from the `GameFile.Item`s which describe them. Having done this, you should find that most of the tests in `AdventureGameTests` now pass.

Activity 3: Secret Buttons (30 minutes)

The aim of this activity is to extend the adventure game with a new `SecretButton` class which implements `Item`. Each `SecretButton` should have a secret "code" so that, when it is pushed, any doors in the same room that have the same code are unlocked. `SecretButton`s are described as follows in the `GameFile` format:

```
Room { description: "Dining Room" }
Room { description: "Hall" }
LockedDoor { from: 0, to: 1, code: 123 }
SecretButton { location: 0, code: 123 }
```

This describes a locked door which connects the "Dining Room" with the "Hall". The player should be able to `Unlock` the door by `Press`ing the `SecretButton`.

What to do. You should create a new class called `SecretButton` class which implements `Item`. You should also make a small modification to the `AdventureGame` class in order to create instances of your `SecretButton` class from the `GameFile.Item`s which describe them. Having done this, you should find that all of the tests in `AdventureGameTests` now pass.

Submission

Your lab solution should be submitted electronically via the *online submission system*, linked from the course homepage. The required files are:

```
swen221/adventure/AdventureGame.java
swen221/adventure/model/Coin.java
swen221/adventure/model/Door.java
swen221/adventure/model/Item.java
swen221/adventure/model/Player.java
swen221/adventure/model/Room.java
swen221/adventure/model/Key.java
swen221/adventure/model/Obelisk.java
swen221/adventure/model/PickupableItem.java
swen221/adventure/util/GameFile.java
swen221/adventure/view/GraphicalUserInterface.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All JUnit test files supplied for the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. This does not prohibit you from adding new tests, as you can still create additional JUnit test files. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.