KIT101 Programming Fundamentals

PP Task 6.1 Objects as Records

Overview

Purpose: Learn how to define your own classes of objects to manage data in your programs.

Task: Create a program that stores data about an entity in a new kind of object, and then displays it to verify that it works.

Learning Outcomes: 2 🔀 3 ✏

Time: Aim to complete this task before the start of Week 7. You may wish to do it in parallel with 6.2CR *Objects with More Abilities*

Resources:

- Introductory Programming Notes:
  - 08 Making Decisions
  - 09 Repeating Actions with Loops
  - 11 Creating Your Own Data Types
- Online Mini-lectures:
  - 📹 Creating Your Own Classes

Note: The next task, 6.2CR Objects with More Abilities, builds on this one. We recommend you complete this one first, getting some feedback from your tutor as you go, and then proceed with 6.2CR even if you haven't yet had this task formally assessed. While this may mean making some of the same changes in both that won't take you long, as 6.2CR is additive.

Submission Details

Upload the following to the MyLO submission folder for this task:

- Program **source code** implementing a new class and a program to interact with the user to create an object of that class (two .java files)
- A **screenshot** showing the execution of your program

Assessment Criteria

A ▩ Completed submission will:

- Have two sources files, one containing main() and other methods for interacting with the user, and the other defining the new data-oriented class, with functionality placed in the appropriate source file
- Have a suitable name (*not* MainProgram) for the source file containing main()
- Apply the necessary validation rules for your chosen option
- Follow the unit's coding style for layout, variable names (use of case), and commenting (including your name at the top)
- Compile and run, with the screenshot showing that it works

Instructions

In this task you will construct possibly your first multi-source file program. While you have implicitly been writing multi-file programs whenever you use another class (Turtle, Scanner, Color) in this one you will have responsibility for all the parts.

You will implement *one* of five programs. Each will consist of:

- a file defining a new class, representing a new data type; and
- a file containing main() and other methods for interacting with the user (this is the program you will actually run).

At a high level your program will look like this (but the names of your files will be tailored to suit your chosen option and with a lot more comments!):

MainProgram.java

```java
import java.util.Scanner;

public class MainProgram {

    //method(s) for reading and validating
    //user input and creating a new DataType

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DataType data;

        //calls method to read details from user
        //and assigns result to data

        //displays data using println
```

DataType.java

```java
public class DataType {
    //variable declarations

    //a constructor

    //getters and setters

    public String toString() {
        //creates and returns a String
        //(does _not_ call println)
    }

}
```

MainProgram.java	*DataType*.java

`    }`
`}`
	Note: Your program will *not* be called MainProgram and your new class type will *not* be called DataType. Both will have names based on the option you choose below.

Follow these general instructions to implement your chosen program. The names of files are given under each option.

Note: You *will* need to refer to the relevant notes to see how to implement your new data type. Those instructions are not repeated here.

1. Create a new program (a source file containing a main() method) and have it import `java.util.Scanner` at the top. Give your program a meaningful name based on your selected option.
2. Create a new class to represent your new data type (this will be in its own .java file):
   a. Declare the instance variables (properties) at the top, making each one as `private` (as in `private int a; //represents something`).

   b. Define a constructor that has one parameter for each instance variable; the body of the constructor should copy the parameter values into their corresponding instance variables.

   c. Define getters and setters for each property, following the templates in the notes.

   d. Define a toString() method that displays the data in the object as specified in your chosen option. The implementation details of this vary depending on the option you choose. Remember that you can create a String value one piece at a time and use += to append to it.
3. Add some *temporary* code in the main program that declares then instantiates (with `new`) an object of your new data type with hard-coded values, and then display the object. This can verify that the constructor and toString() work as intended.

   Note: DrJava will attempt to run whichever file is currently open for editing, so select your source file containing main() before clicking Run. You cannot run your data class because it doesn't contain a main() method.

   Once you've verified it works, delete or comment out the temporary testing code.

4. In the source file for your main program, create the method to read in the necessary details to create your new data object. This method will have a single parameter of type Scanner so that main() can provide it with the Scanner to read from the user. This method will read in data from the user (storing it in local variables) and then create a `new` instance of your data class and return it to main().

5. In that read data method, before you create the new object, you will need to validate one or more user inputs. Use a **while** loop for this: show an error message *while* the user has not entered a valid value, and continue to ask them to enter the data until it is valid. Using the names from the example high-level description above, the general structure of this method will be:

```
public static DataType readDataType(Scanner in) {
    //Local variables for each property of DataType

    //1. Interaction with the user to read valid values into those (may involve a while loop)
    //2. Creation of a new instance of DataType by passing those variables to its constructor
    return /* the new object */;
}
```

Hint: If you completed 4.3CR *User Input Functions* then consider copying some of those methods into your main program and using them to assist in validating the user's input.

6. In main(), call your read data method and store the result in a variable of your new data type, then display that variable using `System.out.println()`.

7. That should make your program complete. Be sure to test it to make sure that it works as expected, then **take a screenshot** and submit the two source files to the submission folder on MyLO.

The five different programs you can choose from are:

1. Cost Calculator: reads in and prints expense data.

2. Sensor Readings: reads in and prints values from sensors.

3. High Scores: reads in and prints values for a high score table.

4. Casting Agent's Talent List: read in film actors for your casting agent to recruit.

5. Make your own…

Tip: Click on the headings for the options you don't choose to collapse those sections, allowing you to see the general instructions and specific details for your chosen option.

Option 1: Cost Calculator

In this task you will start to create a program that will calculate the costs of aspects of a project. The program will allow the user to enter details for an expense, and then display its details.

This program will use an **Expense** class, which you will create, to store and work with expense data. Each **Expense** has a **description** (a String), an **expense id** (an `int`) and a **cost** (an `int`). These become its instance variables.

The program must include the following:

- In `Expense`, instance variable declarations, a constructor to initialise their values, getters and setters for each, and a toString method.
- In the main program, a **readExpense** method that reads in an expense's details from the user, instantiates a new `Expense` object using those details and returns that object to the calling code in main(). Before creating the Expense object it must ensure that the user entered cost is *not* a negative value (so it is positive or 0), asking the user to re-enter the value while not valid.
- `Expense`'s toString() method should return a formatted String according to the following pattern, where *italicised* values refer to those variables in the object:

  "item *id*: *description*, $*cost*" followed by "! justification required" if the cost is larger than 500, or "!! authorisation required" if cost is larger than 5000

  For example, a particular Expense object may appear as:

  ```
  item 234: Hellicopter ride, $6250 !! authorisation required
  ```

- main() should declare an Expense variable, and then perform the following steps:
  1. Print the message "Enter Expense Data:"
  2. Use **readExpense** to read in an Expense from the user, and store it in the local variable.
  3. Print the message "You Entered:"
  4. Print the Expense using `System.out.println()` (it does *not* need to manually call toString).

Note: Remember you only need to do one of these options.

Option 2: Sensor Readings

In this task you will start to create a program that will read values from physical sensors (temperature, humidity, light, etc.) which could then be used to control other devices. At this stage, the program will allow the user to enter sensor details manually, and will print the values back to the screen.

This program will use a **Reading** class, which you will create, to store and work with sensor readings. Each **Reading** has an **annotation** (a String), a **reading id** (an `int`) and and a **value** (a `double`). These become its instance variables.

The program must include the following:

- In `Reading`, instance variable declarations, a constructor to initialise their values, getters and setters for each, and a toString method.
- In the main program, A **readReading** method that reads in a sensor reading from the user, instantiates a new `Reading` object using those details and returns that object to the calling code in main(). Before creating the Reading object it must ensure there is a meaningful annotation provided (its length is at least 15), asking the user to re-enter the value while not valid.
- `Reading`'s toString() method should return a formatted String according to the following pattern, where *italicised* values refer to those variables in the object:

  "*id*: *value*, *annotation*" followed by "(check sensor)" if the value is 0, or "(check reading)" if value is larger than 100

  For example, a particular Reading object may appear as:

  ```
  1138: 0, my shiny internet-connected toaster (check sensor)
  ```

- main() should declare a Reading variable, and then perform the following steps:
  1. Print the message "Enter Reading Data:"
  2. Use **readReading** to read in details from the user to instantiate a Reading object, and then store it in the local variable.
  3. Print the message "You Entered:"
  4. Print the Reading using `System.out.println()` (it does *not* need to manually call toString).

Note: Remember you only need to do one of these options.

Option 3: High Scores

In this task you will start to create a program that will store game scores that could be used to keep track of score ranks. At this stage the program will allow the user to enter game details manually, and it will then print these values back to the screen.

This program will use a **GameScore** class, which you will create, to store and work with game scores. Each **GameScore** has a **user name** (a String), a **game id** (an `int`), and a **score** (an `int`). These become its instance variables.

The program must include the following:

- In `GameScore`, instance variable declarations, a constructor to initialise their values, getters and setters for each, and a toString method.
- In the main program, a **readGameScore** method that reads in the game score data from the user, instantiates a new `GameScore` object using those details and returns that object to the calling code in main(). Before creating the GameScore object, it must ensure that the score is 0 or larger, asking the user to re-enter the value while not valid.
- `GameScore`'s toString() method should return a formatted String according to the following pattern, where *italicised* values refer to those variables in the object:

  "*user name* - *score* (game *game id*)" followed by "n00b" if the value is less than 1000, or "Godlike" if value is larger than 999999

  For example, a particular GameScore object may appear as:

  ```
  bajo - 987 (game 256) n00b
  ```

- main() should declare a GameScore variable, and then perform the following steps:
  1. Print the message "Enter Score Data:"

  2. Use **readGameScore** to read in details from the user to instantiate a GameScore object, and then store it in the local variable.

  3. Print the message "You Entered:"

  4. Print the GameScore using `System.out.println()` (it does *not* need to manually call toString).

Note: Remember you only need to do one of these options.

Option 4: Casting Agent's Talent List

In this task you will start to create a program that will store details of film actors (i.e., stars) being considered by a casting agent. At this stage the program will allow the user to record an actor manually via the console, and it will then print these details back to the screen.

This program will use an **Actor** class, which you will create, to store and work with the actors. Each **Actor** has a *name* (a String), a Screen Actors Guild **membership id** (an `int`), and an estimated **revenue** (an `int`), which is how much they may be worth to a film's gross (in millions) if they are cast. These become its instance variables.

The program must include the following:

- In `Actor`, instance variable declarations, a constructor to initialise their values, getters and setters for each, and a toString method.
- In the main program, a **readActor** method that reads in the actor's details from the user, instantiates a new `Actor` object using those details and returns that object to the calling code in main(). It must ensure that the revenue is 0 or larger, asking the user to re-enter the revenue while not valid.
- `Actor`'s toString() method should return a formatted String according to the following pattern, where *italicised* values refer to those variables in the object:

  "*name* [#*id*] $*revenue*M expected" followed by "(Sharknado only)" if the revenue is less than 5, or "(A-lister)" if revenue is larger than 50

  For example, a particular Actor object may appear as:

  ```
  Hugh Jackman [#24601] $72M expected (A-lister)
  ```

- main() should declare an Actor variable, and then perform the following steps:
  1. Print the message "Enter Actor Data:"
  2. Use **readActor** to read in details from the user to instantiate an Actor object, and then store it in the local variable.
  3. Print the message "You Entered:"
  4. Print the Actor using `System.out.println()` (it does *not* need to manually call toString).

Note: Remember you only need to do one of these options.

Option 5: Make your own...

Use the above four options to guide you to create your own program.

It must have the same features:

- A new class type with at least 3 fields, using a mix of types: `String`, `int` and `double`.
- The class must have a constructor, getters, setters and a toString() that generates a formatted String of the data it contains. The result of toString must have at least two alternative forms based on values within the object.

- A method in the main program to get details from the user, validate at least one of the pieces of information and then instantiate an object of your new type.
- In main():
    - declare a variable of the new type
    - call the read method get an object you can assign to the variable
    - display the object

Additionally, if you are planning to proceed to 6.2CR: think of some additional attribute for your new type that could be modelled as an *enumerated* type with at least three values.

Tip: Do not spend too long thinking about this...
