CSC1016S Assignment 4

Simple Java Classes and Junit

Introduction

This assignment concerns reinforcing OO concepts through creating and manipulating types of object; and the writing of simple class declarations.

Key points are that an OO programmer (i) often uses predefined program components i.e. classes, (ii) often develops program components, not whole programs and (iii) needs techniques and tools for checking/evaluating their work.

Exercise one involves predefined classes. Your task is to develop a suite of Junit tests to demonstrate those classes function correctly.

Furthermore**,** in future assignments, your solutions will be evaluated for correctness and for the following qualities:

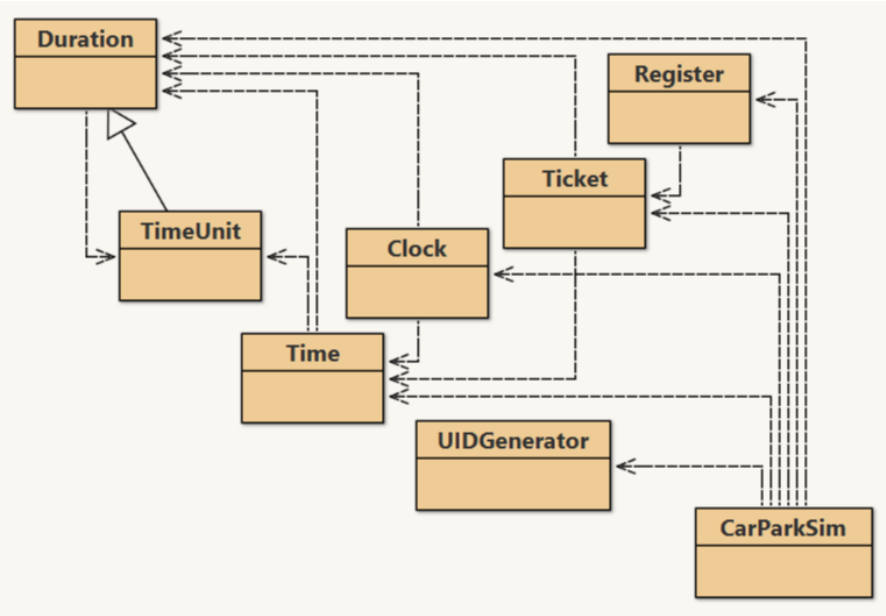
- The use of object types, object creation, and the reading and writing of object fields.
- Documentation
    - Use of comments at the top of your code to identify program purpose, author and date.
    - Use of comments within your code to explain each non-obvious functional unit of code.
- General style/readability
    - The use of meaningful names for variables and functions.
- Algorithmic qualities
    - Efficiency, simplicity

These criteria will be manually assessed by tutors and commented upon. The number of marks to be deducted for non-compliance with the above will be communicated in each assignment in future.

The Scenario

This exercise is themed (to be continued in the next assignment). It concerns modelling aspects of a pay-to-stay car. The kind of car park in question has a ticket machine at the entrance and a cashier at the exit. A driver, on entering the car park receives a ticket stamped with the arrival time. (The arrival time is also recorded on the magnetic strip on the back.) On exit, the driver gives the ticket to the cashier, the duration of the stay is calculated and from that, how much must be paid.

Here is the design:



On the assignment page you will find Time, Duration, Clock, and TimeUnit.

An arrow from a class A to a class B indicates that A uses B in some way.

A Clock object is used to simulate time and the passing of time. Basically, it stores a time value that can be advanced. It may be found on the Amathuba page for this assignment.

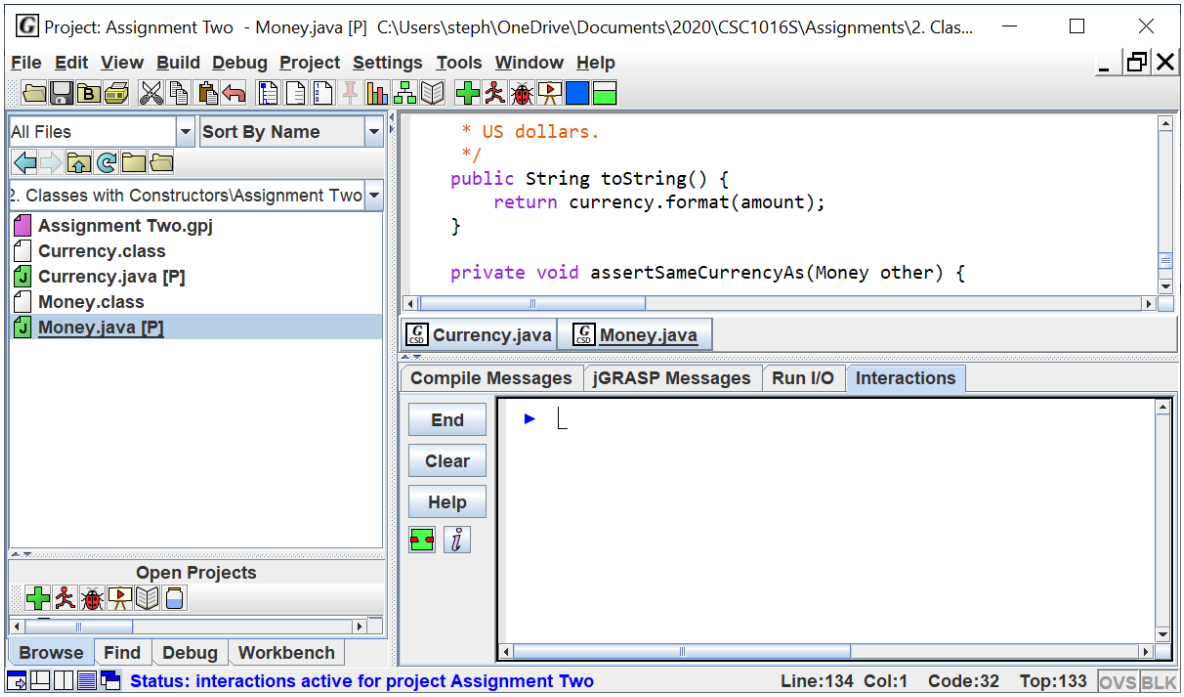
The Time and Duration classes are essentially the same as used in assignment 1. We have tweaked them a bit and have created a new TimeUnit class. You should use these versions for this assignment. (The big change is the addition of some string formatting for Duration.)

JGrasp Interactive feature

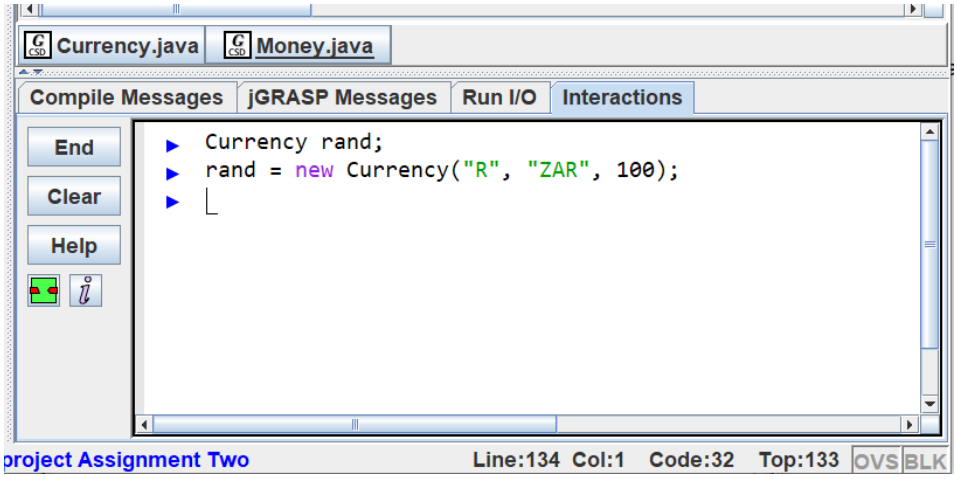
A useful tool evaluating work on the fly and for experimenting with Java is the JGrasp interactive feature. This is a part of JGrasp that allows you to enter Java statements and expressions and have them evaluated in much the same way as the Python Shell in the Wing 101 IDE (used in CSC1015F).

To illustrate, question one involves the use of a Money class and a Currency class. Let's say that we wish to confirm our understanding of these classes by experimenting with creating and manipulating Money and Currency objects; specifically, we would like to see if we can devise the correct sequence of statements to add the sums R25 and R16.50 together.

Assuming we have created a new JGrasp project for the assignment, have added the Money.java and Currency.java files from the Amathuba assignment page, and compiled them, we start by clicking on the 'Interactions' tab.

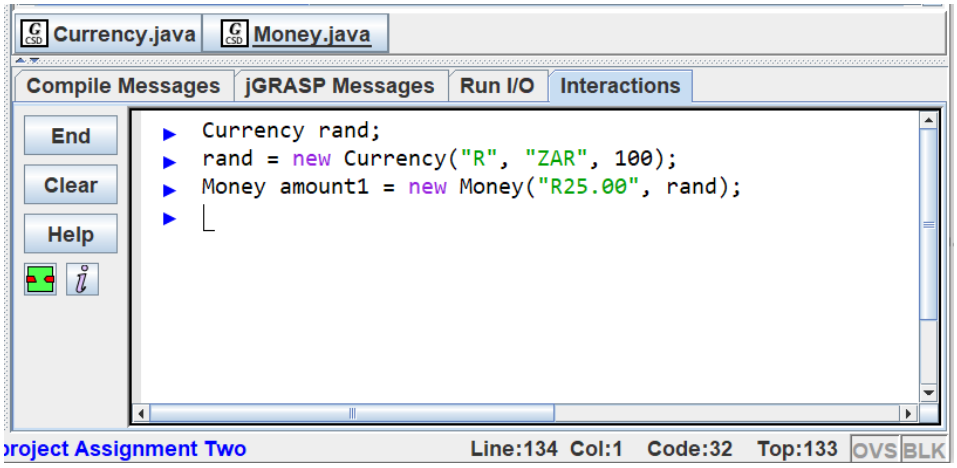


We can then enter our statements. We start by creating a Currency object to represent South Africa Rand:



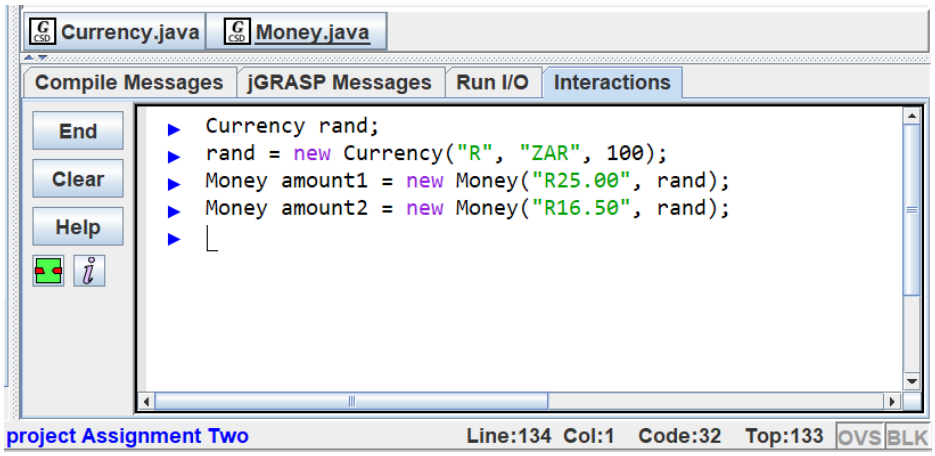
We declare a variable called ‘rand’ that will store a Currency. Then we create a new Currency object with the actual parameter values “R”, “ZAR” and 100 and assign it to the variable. (‘R’ is the symbol used for South African Rand amounts, ‘ZAR’ is the ISO 4217 code for the South African Rand, 100 is the number of minor units (cents) in a Rand.)

Now that we’ve made a Currency object, we can create a Money object representing R25.

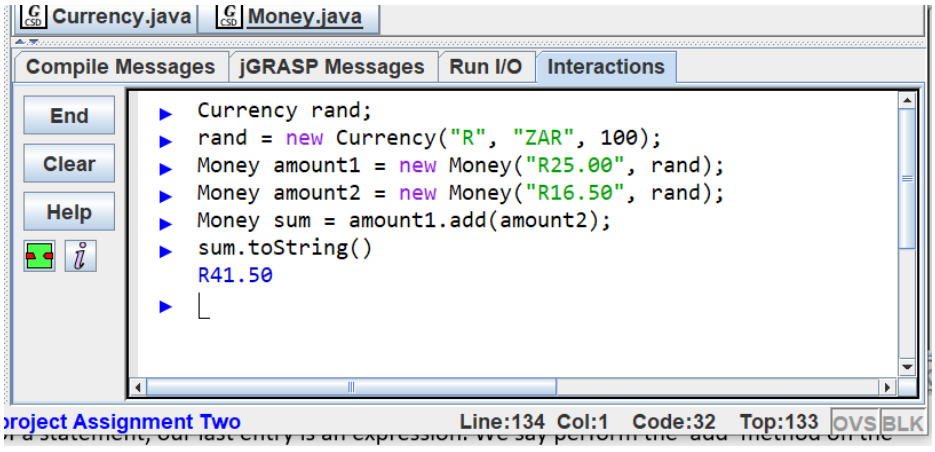


In this case we've declared a variable, created an object and performed an assignment all in one statement. The variable is called 'amount' and stores a Money value. The Money object is created with the actual parameter values "R25.00" and 'rand' the name of the variable referring to our Currency object.

We can create a second Money object representing R16.50.



We've assigned it to a variable called 'amount2'. And now we can try adding the amounts together.



We declare a final variable called 'sum' and assign to it the result of adding the two amounts.

We perform the 'add' method on the Money object referred to by 'amount1', passing the Money object referred to by 'amount2' as a parameter. The result of the addition is a new Money object.

Our final entry is not a statement but an expression, we perform the 'toString' method on the Money object referred to by 'sum'. The result is a string representation of the amount. Since we haven't tried to assign it to anything, JGrasp just prints it out.

Exercise One [40 marks]

Unit testing

Unit testing is an automatic testing technique for individual modules of a large software system. Tests are defined and executed automatically to rapidly test and re-test a class during development without user intervention. This helps to scale up testing when a program gets very large. It also ensures the correctness of each individual class within the system.

JUnit is one of the most popular unit testing frameworks for Java. The advantage of a framework (rather than writing small programs to test each class) is that there is a standard API, common understanding of the testing approach among Java programmers and integrated support in many IDEs.

Time and Duration classes

Study the following specifications for Time and Duration types of object.

Class Time

A Time object represents a twenty-four-hour clock reading composed of hours, minutes and seconds.

Constructors

Time(String reading)

// Create a Time object from a string representation of a twenty-four-hour clock reading
// of the form 'hh:mm[:ss]' e.g. "03:25", "17:55:05".

Methods

public Duration subtract(Time other)

// Obtain the Time that results from subtracting the given period from this Time.

public String toString()

// Obtain a String representation of this Time.

Class Duration

A Duration object represents a length of time (with millisecond accuracy).

Methods

public long intValue(String timeunit)

// Obtain an integer value that represents as much of this duration object that can be expressed
// as a multiple of the given time unit
// For example, given a duration object d that represents 1 hour, 4 minutes, and 30 seconds,
// d.intValue("minute") produces 64.
// Permissible time units are: "millisecond", "second", "minute", "hour"..

For the Time class we've listed one way of creating it: a Time object can be created from a string representing a particular 24 hour clock time. The following code snippet provides an example:

```
Time t = new Time("13:45");
```

We've given two methods of manipulating a Time object, one is called '`subtract`' and the other '`toString`'. The subtraction method provides a means for subtracting one time from another to

obtain a duration i.e. obtaining the period between them. The other method provides a means of obtaining a string representation of the object i.e. something printable.

Say we had two Time objects referred to as '`t1`' and '`t2`', we could subtract `t2` from `t1` using the expression '`t1.subtract(t2)`'

The `subtract` method returns a value that is a Duration object.

We've given you one method for Duration objects, '`intValue`'. The method accepts a String parameter which must be the name of a time unit. It returns an integer representing the number of that unit closest to the duration value. An example is given in the specification: given a duration object, `d`, that represents 1 hour, 4 minutes, and 30 seconds, the expression `d.intValue("minute")` evaluates to 64.

Task

Construct a Junit test class called `TestOfTime` that contains the following tests of the Time and Duration classes:

Test Number	Class Name	Test purpose
1.	Time	Check that a Time object does actually store the time value provided as a parameter during creation (by calling toString).
2.	Time	Check that subtracting an earlier Time from a later Time produces a Duration of the correct length.
3.	Time	Check that subtracting a Time from itself produces a zero Duration.
4.	Duration	Check that the 'intValue' method works with a parameter of "millisecond".
5.	Duration	Check that the 'intValue' method works with a parameter of "second".
6.	Duration	Check that the 'intValue' method works with a parameter of 'minute'.
7.	Duration	Check that the 'intValue' method works with a parameter of 'hour'.

Download the JUnit packages from ***Content>>Resources>>Software>>Java SDK>>Junit*** from Amahuba.

JGrasp has built-in integration with JUnit.

- Go to Tools/JUnit and use Configure to first set the location of the JUnit packages you downloaded. Then there will be options to run JUnit tests from within JGrasp.
- When you add source files to a JGrasp project, JGrasp will automatically detect which ones are normal source files and which ones are JUnit test classes.

Alternatively, in order to compile your JUnit test class from the command-line, use a command such as:

```
javac -cp junit.jar:hamcrest-core.jar:. TestOfTime.java
```

And in order to execute your JUnit tests from the command-line, use a command such as:

```
java -cp junit.jar:hamcrest-core.jar:. org.junit.runner.JUnitCore
TestOfTime
```

Exercise Two [30 marks]

This exercise involves the use of the `Item` and `ShoppingCart` classes of an object that maybe described as follows:

`Class Item`

An Item object represents a product being purchased and it is composed of product name, unit price and quantity.

Constructors

```
public Item(String productName, int quantity, double unitPrice)
```

// Create an Item object from a given item purchased. An item can be either clothing or

// groceries. An item has attributes product name, quantity and unit price.

Methods

```
public String getProductName()
```

// returns the name of the item purchased

```
public double getUnitPrice()
```

// returns the unit price of an item purchased

```
public int getQuantity()
```

// returns the number of items purchased.

`Class ShoppingCart`

A ShoppingCart object represents the total amount to be paid at checkout. This includes the actual amount due, applicable value added tax and discount (if any).

Constructors

```
public ShoppingCart()
```

// Create a ShoppingCart object from an item purchased with the following attributes;

// total cost of items, applicable discount, applicable value added tax and the actual

// amount to be paid at checkout.

Methods

```
public void addItems(Item item)
```

// add items to the shopping cart

```
public void deleteItems(Item itemToDelete)
```

// remove any item(s) not needed from the shopping cart

```
public void queryCart()
```

// display all items in the shopping cart

```
public double getTotalAmount()
```

// calculate the total amount of all items in the shopping cart

```
public void getDiscount(String coupon)
```

// calculate an applicable discount on the items purchased based on the

//discount coupon you have.

```
public double getPayableAmount()
```

// amount paid at checkout when shopping has been completed

```
public void printInvoice()
```

// print the invoice (receipt) of all items purchased

Your Task: on the Amathuba page for the assignment, you will find the `Item` and `ShoppingCart` classes. Download these and use them to write a class called "`TestShoppingCart`" that has the main method to allow the user to capture details of item(s) purchased and print out an invoice for the customer.

Sample I/O (*NOTE:** *the input from the user has been shown in **bold):

```
How many items would you like to add to your Shopping Cart?:
0
Your Shopping Cart is empty.
```

Sample I/O (*NOTE:** *the input from the user has been shown in **bold):

```
How many items would you like to add to your Shopping Cart?:
2
Enter the Product Name:
Milk
Enter the Quantity:
16
Enter the Unit Cost:
16.99
Enter the Product Name:
Eggs
Enter the Quantity:
90
Enter the Unit Cost:
1.29
The Shopping Cart has the following items:
Milk: 16
Eggs: 90
--- Shopping Cart with all items ---
Milk 16 16.99 271.84
Eggs 90 1.29 116.10
  Total    :387.94
  Discount :77.59
  Tax      :43.45
  Total    :353.80
```

Exercise Three [30 marks]

This question concerns (i) writing a `Student` class to model a student's name, composed of a first name, middle name and last name, and (ii) using a supplied test suite to drive the process.

The `Student` class should meet the following specification:

Class Student

A Student object represents a student. A student has a first name, middle name and last name.

Methods

public void setNames(String first, String middle, String last)

// Set the first, middle and last names of this Student object.

public String getFullName()

// Obtain the full name of this Student with the middle name converted to an initial only.

Thinking of the problem in terms of testing, there are three requirements that must be met:

A. The `setNames()` method sets the names stored in the object i.e. the values of `firstName`, `middleName`, `lastName`.
B. The getFullName() method obtains the name of the student with the middle name converted to an initial only.
C. The getFullName() method does not change anything – i.e. it does not modify the values of `firstName`, `middleName`, `lastName`.

Note that, these requirements are interdependent i.e. you can only check `setNames()` works by using `getFullName()`, and you can only check `getFullName()` works by first using `setNames()`.

On the Amathuba page for this assignment, you will find a program called `TestStudent` that you should add to your JGrasp project for this question.

The program assumes that you 've constructed a `Student` class, and as its name suggests, it runs tests on Student objects.

It codes test requirements as a set of two tests: the first test evaluates A and B in conjunction, and the second test focuses on C.

We could have written one test covering all three requirements, however, two tests offer a more precise diagnosis of problems with code.

Use the `TestStudent` program to develop your `Student` class:

- If a test fails, look at the TestStudent program to see what the test does.
- If the tests pass, then your Student class will receive full marks from the automatic marker.

HINT: To start your Student class, we suggest using the following instance variables:

```
Instance variables
private String firstName;
private String middleName;
private String lastName;
```

Submission

Submit the `TestOfTime.java`, `TestShoppingCart.java`, and `Student.java` source files to the automatic marker in a ZIP file bearing your student number.

Appendices

Java Arrays

Arrays in Java bear a lot of similarity to Python Lists. Given an array, `A`, and an index value, `i`, a value, `v`, can be stored with the expression "`A[i]=v`". Similarly, a value can be retrieved, with the expression "`A[i]`" e.g. retrieving *v* and storing in a variable *n* is written "`n=A[i]`".

The differences are:

- An array stores a TYPE of value, which must be given when declared/described.
- An array is a type of object, and as such, is created using 'new'.
- An array has a fixed size which must be given when created.

Assume the following BMI class:

```
public class BMI {
	double height;
	int weight;

	double calculateBMI() {
		return weight/(height * height);
	}
}
```

Let's say we want an array that holds ten BMI objects. The following code snippet (i) declares a variable that can store an array of BMI, then (ii) creates such an array and assigns it to the variable:

```
//...
BMI[] records;
records = new BMI[10];
// ...
```

The variable declaration looks similar to others that we've used. The type is "`BMI[]`". It's the brackets that indicate the variable can store an array that stores BMI objects. Without the brackets, of course, it would just be a variable that can store a BMI object.

The creation expression is similar. The type of thing being created, "`BMI[10]`", is an array that can store BMI objects, the size of the array is ten.

Initially the array does contain any BMI objects. (The value at each index is the special value '`null`'.) Extending the code snippet as follows, we create a BMI object and insert it at location zero:

```
//...
BMI[] records;
records = new BMI[10];

BMI bmi_record = new BMI();
bmi_record.height = 165;
bmi_record.weight = 57;
records[0] = bmi_record;

System.out.println(records[0].height);
System.out.println(records[0].weight);
System.out.println(records[0].calculateBMI());
//...
```

The snippet ends with print statements. Each accesses the BMI object at location zero, i.e. this is what the expression "`records[0]`" does, and then one of the object's components. The first print accesses the `height` field, the second the `weight` field, and the third the `calculateBMI()` method.

For completeness, consider the following additional statements:

```
//...
System.out.println(records[0]);
System.out.println(records[1]);
//...
```

You might be inclined to think that the first statement prints out the BMI object stored at location zero, i.e. the height and weight. In fact, it prints something like the following:

```
BMI@7e6c04
```

The output consists of the name of the type of object (BMI) followed by an '@' sign, followed by what's called a "hashcode", a kind of identity code, and which we won't get into here. (If we had another BMI object and tried to print that we would generally get a different hashcode for it.)

The second print statement will output the following, since we haven't stored a BMI object at that location:

```
null
```

Finally, given an array, `A`, we can obtain its length with the expression "`A.length`".