Assignment 1

CompSci 230 S2 2015

Clark Thomborson

Submission deadline: 4pm, Friday 27 March 2015

Version 1.02 of 16 March 2015: corrected my reasoning about payrises in Q7; added bullet-lists, subheadings, and bold-faced phrases to highlight the submission requirements in all questions.

Total marks on this assignment: 40. This assignment counts 4% of total marks in COMPSCI 230.

Learning goals. Basic level of competency in the Java programming language, the Eclipse IDE, and OO design.

Getting help. You may get assistance from anyone if

- you become confused by Java syntax or semantics,
- you don't understand anything I have written in this assignment handout,
- you have difficulty setting up or using Eclipse,
- you don't know how to take screenshots, or how to insert these into your assignment submission, or
- you have any other difficulty "getting started" on this assignment.

Working independently. The homework assignments of this course are intended to help you learn some important aspects of software development in Java, however they are also intended to assess your mastery (or competence) of these aspects. You must construct ***your own answer to every question***.

If you get "stuck" on a question, you should diagnose your own difficulty and then do something to resolve it! Perhaps you need to learn some concept, or perhaps you need to learn how to use a tool, or perhaps you need to build up some problem-solving skills?

After you have diagnosed your difficulty, you should look for some appropriate way to resolve it. Perhaps it'd be helpful to work through a lesson in the Java Tutorials, or to ask someone for advice on using Eclipse, or to attempt a series of "simpler" but similar problems?

If you are able to "diagnose" and then "repair" your difficulty with answering one of the questions on this assignment, congratulations! I reckon you'll need to develop and exercise just one more skill, prioritisation, to become a competent professional in any field – including software development. Your prioritisation skills are being tested indirectly in this assignment, because you have many other things you could be doing before the submission deadline! You may be prioritising your learning of concepts or gaining of skills, within Java or in any other domain, which are not directly relevant to this assignment -- but which might help you resolve some future difficulty in your educational, personal, or professional development.

English grammar and spelling. You will not be marked down for grammatical or spelling errors in your submission. However if your meaning is not readily apparent to the marker, then you will lose some marks.

Resource requirements. You will be using the following software in this assignment:

1. An up-to-date version of the Java SE 8 runtime environment (JRE).
2. A recent version (SE7 or SE8) of the Java development kit (JDK) for your computer.
3. A recent version (Luna) of Eclipse for your computer.
4. A word-processing program such as LibreOffice or Microsoft Word.
5. A screenshot utility for your Windows 8, Mac OS X, or Linux computer.
6. The HuckleBuckle v1 codebase – this is available for download on the CompSci230 Assignments webpage.

Items 1-6 are installed on all lab stations in building 303. Other University computers, e.g. at the Kate Edger facility do *not* have all this software. It is all freeware, so you can install it on a home computer or laptop.

7. A binary distribution of ArgoUML v0.34. (Installation instructions are in Part 2 below.)

Submissions. You must submit electronically, using the Assignment Drop Box (https://adb.auckland.ac.nz/). Your submission must have **one document** (in PDF, docx, doc, or odt format) with your written answers to the questions in this assignment, and one jarfile with the source code you developed for the last question.

Handwritten diagrams. The scanners in the computer lab will help you convert any handwritten diagrams into images you can include in your submission. However you may have difficulty using the scanner interface the first time you try, so I'd advise you to scan and email a trial document at least *a day before* the submission deadline.

ADB response time. The ADB system is sometimes unresponsive, when it is under heavy load due to an impending stage-1 assignment deadline. I will extend your submission deadline by up to one hour, if I see any evidence (from the submission logfiles) of ADB overloading. So: please be patient if the ADB seems slow to respond.

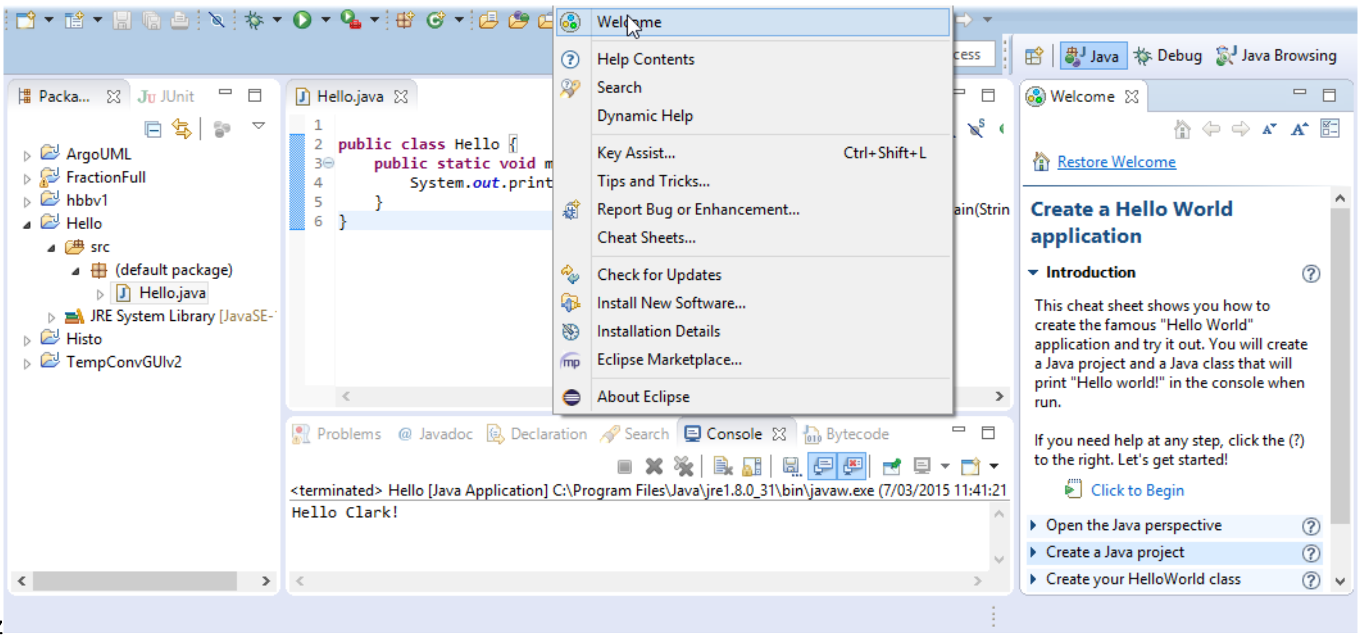
Checking your own submission. I strongly recommend you download a copy of your submission, after you have obtained (and retained!) a submission receipt from the ADB. It'll take you only a few minutes to confirm that you have submitted your latest set of answers to this assignment, and that your jarfile meets all requirements. The error rate on assignment submissions in COMPSCI 230 is about 1%, on a historical basis, because students occasionally submit an assignment from another class or the "wrong version" of a jarfile. This is only 4-sigma quality: is it good enough for you?

Resubmissions. You may resubmit at any time prior to 4pm Wednesday 1 April 2015. However your markers will see only your latest submission – you will lose marks unnecessarily if you make a partial resubmission, so please be careful to submit *all*

Lateness penalties. If you submit or resubmit after the deadline (4pm Friday), you will be penalised 8 marks (20% of possible marks) or 20 marks (50% of possible marks), depending on whether your submission or resubmission was within 72 hours of the deadline. No submissions will be accepted more than 120 hours (5 days) after the deadline. Lateness penalties will not reduce your marks on this assignment below 0.

Part 0: Setting up your IDE

Locate or install the software listed in items 1-5 of the Resource Requirements section. Work through the "Create a Hello World application" tutorial in Eclipse. This tutorial is accessible through the Help/Welcome menu item, as indicated below. You may ask anyone for assistance on this portion of the assignment.



```
public class Hello {
    public static void m
        System.out.print
    }
}
```

```
<terminated> Hello [Java Application] C:\Program Files\Java\jre1.8.0_31\bin\javaw.exe (7/03/2015 11:41:21
Hello Clark!
```

Create a Hello World application

Introduction

This cheat sheet shows you how to create the famous "Hello World" application and try it out. You will create a Java project and a Java class that will print "Hello world!" in the console when run.

If you need help at any step, click the (?) to the right. Let's get started!

z

Part 1: Introductory Exercises with Eclipse and Java

0) **(1 mark)** Create a document in your word processor. Put **your name, student ID number, your UPI, today's date, and an assignment identifier** (e.g. "CompSci 230 Assignment 1") on the first page of your submission.

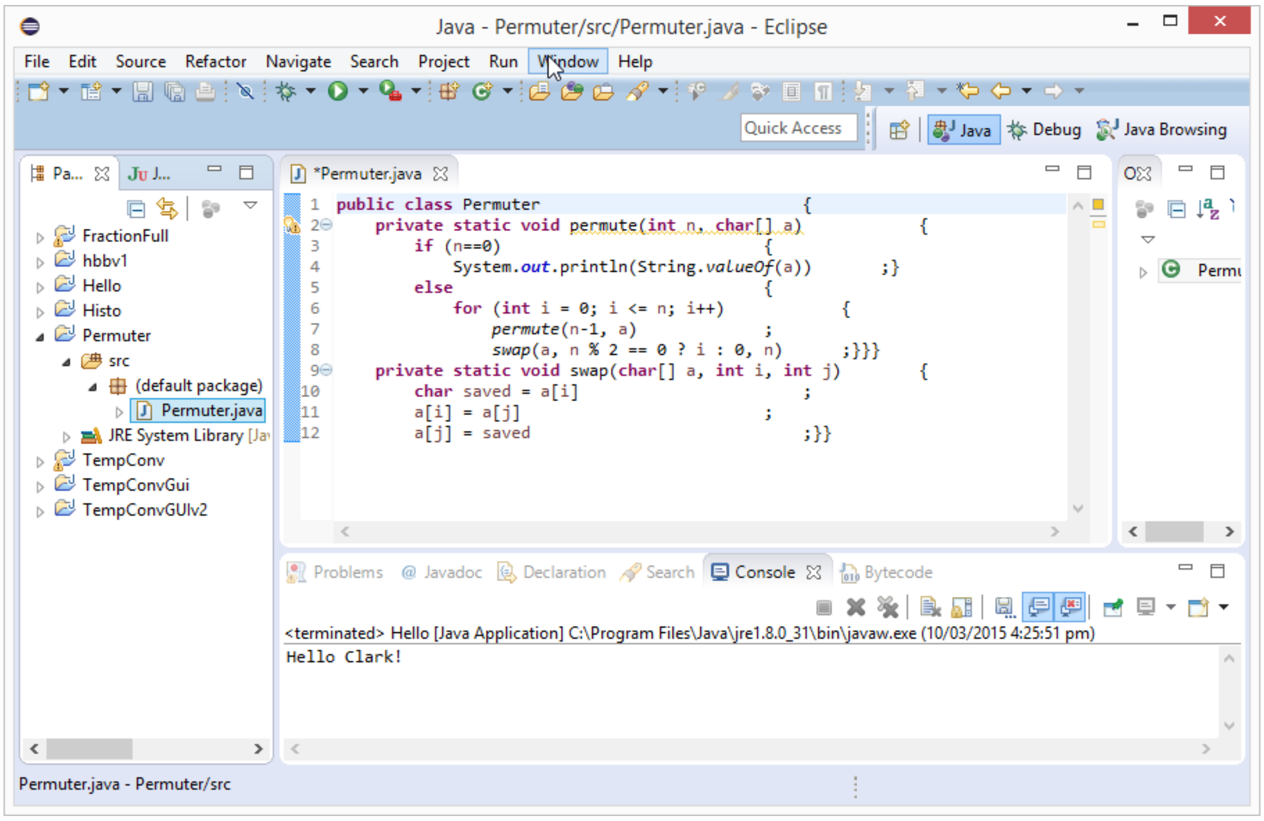
1) **(4 marks)** Modify the "Hello World!" application you created in Part 0 of this assignment, so that your version of this application prints "Hello XXX!" where XXX is your name. Take a screenshot which shows your source code and console output. Paste **this screenshot** into your submission document, indicating that it is your answer to question 1.

2) **(4 marks)** Read the "Source Code Comments" section of the "A Closer Look at the `Hello World!' Application" lesson in the Java Tutorials. Add a ***documentation comment*** to the `Hello.java` class you created in the previous question. Your comment should be similar in style to the example in the Java Tutorials, but it should describe the behaviour of *your* version of `Hello.java`. Cut and paste **the characters (and formatting) of your `Hello.java` file, from your Eclipse window** into your submission document: this is your answer to question 2.

3) **(4 marks)** In Eclipse, create a new Java project called `Permuter`. Next, create a new Java class called `Permuter`. Replace the skeleton implementation of the `Permuter` class (which was generated automatically by Eclipse) with the following (Python formatted ;-) source code.

```
public class Permuter                                             {
      private static void permute(int n, char[] a)                {
             if (n==0)                                            {
                   System.out.println(String.valueOf(a))          ;}
             else                                                 {
                   for (int i = 0; i <= n; i++)                   {
                          permute(n-1, a)                         ;
                          swap(a, n % 2 == 0 ? i : 0, n)          ;}}}
      private static void swap(char[] a, int i, int j)            {
             char saved = a[i]                                    ;
             a[i] = a[j]                                          ;
             a[j] = saved                                         ;}}
```

Your Eclipse display should be similar to the following



```
public class Permuter                              {
    private static void permute(int n, char[] a)            {
        if (n==0)                                  {
            System.out.println(String.valueOf(a))          ;}
        else                                       {
            for (int i = 0; i <= n; i++)         {
                permute(n-1, a)                    ;
                swap(a, n % 2 == 0 ? i : 0, n)       ;}}}
    private static void swap(char[] a, int i, int j)        {
        char saved = a[i]                            ;
        a[i] = a[j]                             ;
        a[j] = saved                             ;}}
```

```
<terminated> Hello [Java Application] C:\Program Files\Java\jre1.8.0_31\bin\javaw.exe (10/03/2015 4:25:51 pm)
Hello Clark!
```

Note that there is one warning message. Read this message, by hovering your mouse over the yellow exclamation mark at the left of the second line in `Permuter.java`. Note that the `permute()` method is not invoked by any method within the `Permuter` class, and (because it is `private`) it cannot be invoked by any method in any other class. Eclipse has issued a warning message to notify the developer that `permute()` has been defined but not used. You'll soon resolve this issue by adding a `main()` method which calls `permute()`.

Please also note that the whitespace (spaces and tabs) in the code at the beginning of this question was *not* reliably reproduced by my cut-and-paste operation. As a result, the code merely looks messy (rather than looking as though it were written by a Python programmer who was making fun of Java's syntax.) I have included this curious example in this assignment because I think it is very important for Python programmers to realise that indentation in a Java program is solely a question of style: there are no syntactic or semantic implications. Java statements are separated by semicolons, as opposed to newline marks in Python. Blocks of Java statements are delimited by curly braces, as opposed to a changed indentation in Python.

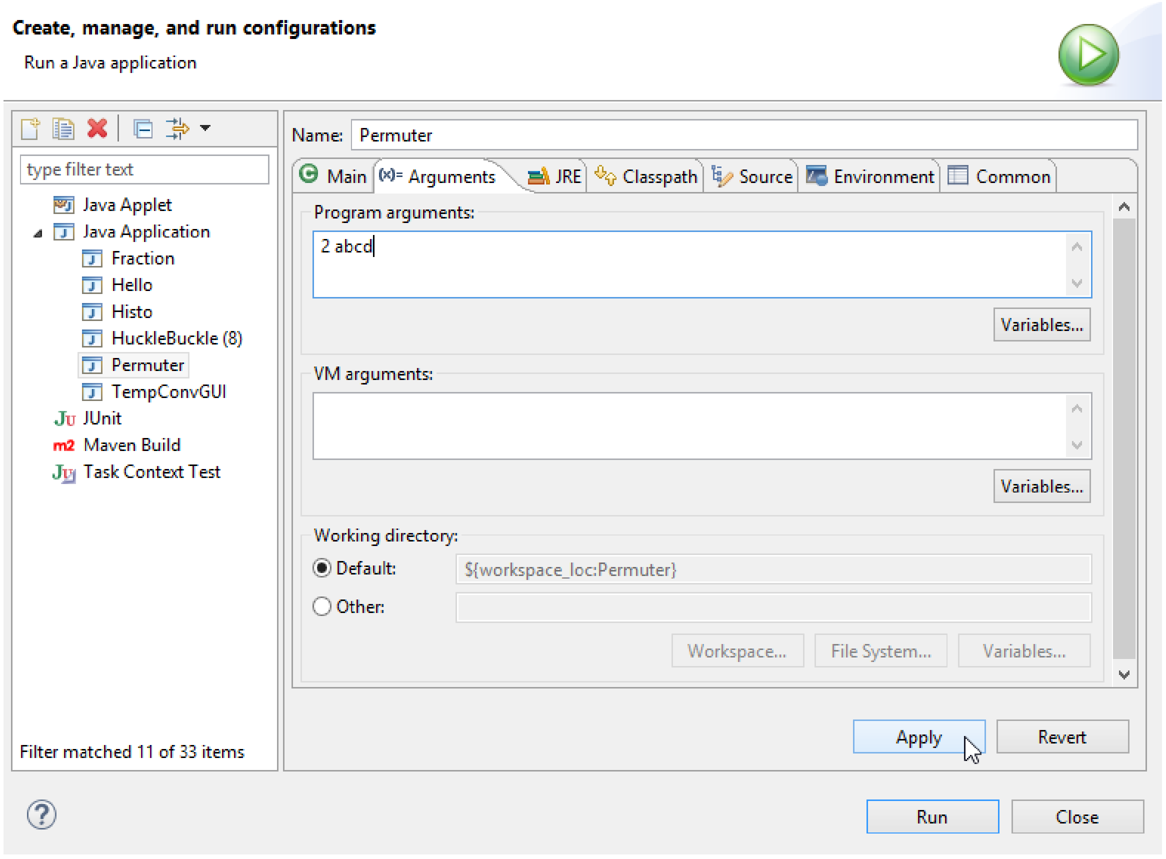
Now use the Source/Format command of Eclipse to "clean up" the whitespace in `Permuter.java`. The source code is now in a common "whitespace style" for Java, making it much easier for other Java programmers to read; but it won't make anyone smile. (Thanks to Eric Cheyne for suggesting this amusing snippet of code!)

Add the following `main()` method to `Permuter.java`. It is now a Java application.

```
public static void main(String[] args){
        permute(Integer.parseInt(args[0]), args[1].toCharArray());
}
```

Compile and run the `Permuter` application. Note: Eclipse will report a run-time error message (an `ArrayIndexOutOfBoundsException`) because you haven't specified any command-line arguments. This `main()` requires two arguments: an integer n and a string of length at least n.

Invoke the "`Run/Run Configurations…`" menu item of Eclipse, then shift to the `Arguments` tab in the middle pane. Type "`2 abcd`" (without the quotes) in the `Program Arguments` text-area. Your display should look like this:



Name: Permuter

Program arguments: 2 abcd

VM arguments:

Working directory:
[✓ Default:] ${workspace_loc:Permuter}
Other:

Now click "`Apply`" (to save these arguments for use in future runs), and then click "`Run`" to see what happens when your `Permuter` application runs with these arguments.

You may receive assistance from anyone to complete the steps above.

a) **For assessment:** cut and paste **the output (from your console window)** into your assignment submission. This text (not a screenshot!) is your answer to question 3a.

b) **For assessment:** Think (briefly) about what would be another interesting set of program arguments for this application. Use the "`Run/Run Configurations…`" dialog of Eclipse to test the `Permuter` application on this new set of arguments. Your answer to question 3b should have three parts:

   i) the program arguments you selected,

   ii) the output or error message your application produced when given these arguments, and

   iii) an English sentence explaining why (or how) you selected these arguments.

   Note: in the Software Quality unit of this course, you will learn some methods for selecting test cases. You are *not* expected to be following any particular testing method in this question, but I will expect you to put a little effort into answering this question, rather than just typing arguments carelessly and observing the outputs.

4) **(4 marks)** *Initial steps:* Look at the table of contents of the Language Basics trail, then review (or read) the "What is an Object?" and "What is a Class?" lessons in the Java Tutorials.

Next, skim through the Variables and Operators lessons, to confirm that you already know these semantic concepts (but probably in a different syntax!) from your introductory-programming course. You shouldn't attempt to learn all of this syntax at this time. Your goal, instead, should be to get some ideas of where you should look for help, when (not if!) you make a syntax error when writing simple Java statements.

Start reading Section 3.4 of java4python. You will find a 4-line program in Python which has been translated into a 19-line `TempConv.java` file. You will also find a brief explanation of three important concepts that are illustrated by this program: importing packages, declaring variables, and reading console input. Later in this course, you'll learn a bit more about these concepts. For now, you should gain a rough understanding of these concepts, so that if someone says "console input" you have some idea of what this phrase means, and that you'd have some idea of where to look if you needed to know enough about it to implement it!

Now create a `TempConv` project in Eclipse, and paste a copy of the 19-line `TempConv.java` from the java4python into a `TempConv.java` file. Run this application. Note: you'll have to mouse-click in the Console area, before you'll be able to provide any console input to this application.

You may receive assistance from anyone to complete the initial steps in this, and any other, question on this assignment.

For assessment:

a) Type an input to `TempConv`. In your submission document, and submit **the input you typed and the output you observed**.

   Modify the second `println()` statement, replacing `cel` by the following method-call

   ```
   String.format("%.1f", cel)
   ```

   After this change, the `TempConv` application no longer uses the default method, `Double.toString()`, when it converts the value of `cel` to a `String` that is printable at the console.

Note: I do not expect you to study the format() method of the `String` class. This is fairly complicated syntax, which is not examinable. However it is of great importance to format output carefully when you're designing usable software, so I want you to know of the existence of this method – even though you may never become proficient in its use.

b) Test the modified `TempConv` application using your input from part a), and submit **the output you observe**.

5) (**4 marks**) *Initial steps:* Continue reading Section 3.4 of java4python. You'll find the code for a simple Swing app called `TempConvGUI`. Build this application in Eclipse.

Add a documentation comment to your version of the `TempConvGUI` class, by selecting the first line of its source file in the Eclipse editor then invoking the Source/Generate Element Comment feature. This will add an @author tag for use in JavaDoc. Cut and paste the following into this JavaDoc comment, then adjust it so that it shows your name rather than mine. Hint: you can use the Eclipse formatting wizard (Source/Format) to adjust the whitespace after you make your changes.

```
/**
 * GUI which converts a temperature from °F to °C
 * <p>
 * Adapted from http://interactivepython.org/courselib/static/java4python
 *
 * @author Clark Thomborson (ctho065)
 * @version 1.0
 *
 */
```

Note: JavaDoc style is *not* examinable in this course, aside from the elements you see above: a single-line summary, followed by zero or more paragraphs delimited by <p>, the @author tag and the @version tag. However I'd encourage you to read Stephen Colebourne's blog on Javadoc Coding Standards, if you want to get some idea of the stylistic conventions you *might* be expected to follow in some future workplace or open-source development group.

For assessment:

- Add a not-quite-trivial feature to `TempConvGUI`, by adding at most a few lines of code. For example, you might add
  - an indefinite loop (so that the application will convert more than one temperature value), or
  - a test for an input temperature that's below absolute zero (so that the application never prints absurd °C temperatures).
- Update the JavaDoc comment for this class by changing its first line (summary description) or the detailed description (in paragraphs after the summary), and the version number. Because you have added a minor feature, your code should be version 1.1.

Cut and paste all of **the source code from your class** into your submission document (thereby creating a "listing" of your code which your marker will assess), and **add a screenshot (or at most two) to illustrate your new feature**.

Part 2: Running an executable jarfile (ArgoUML)

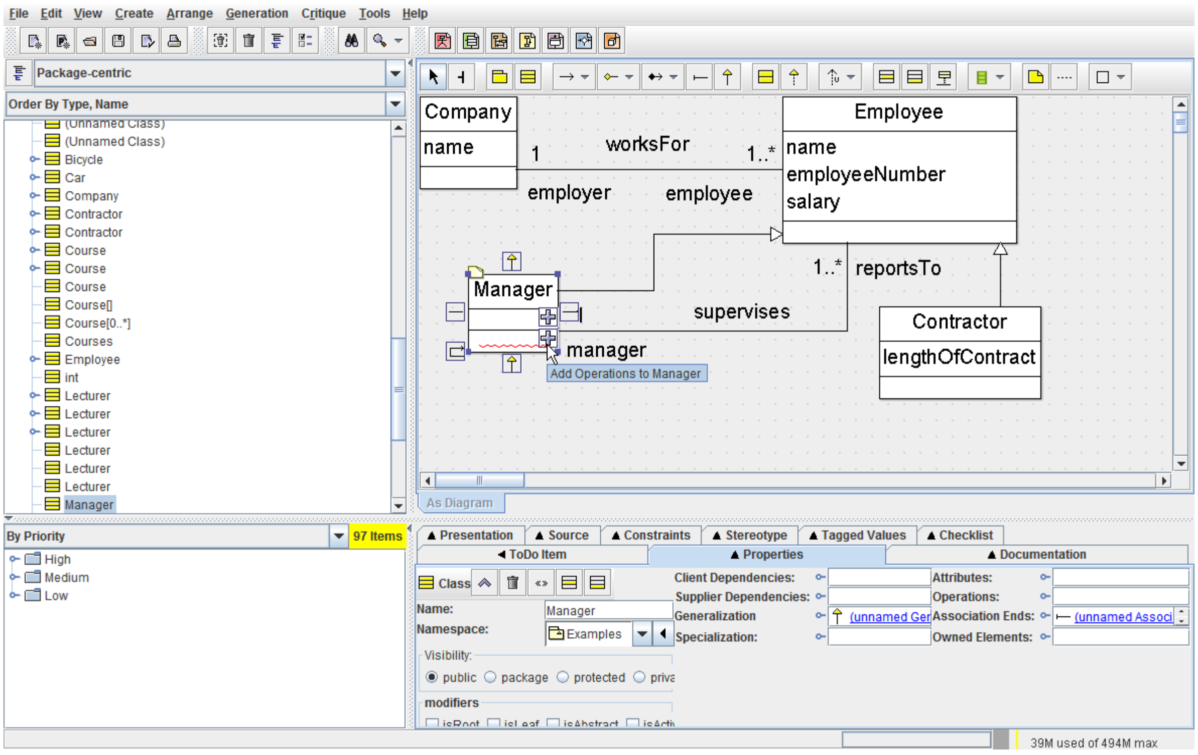
6) **(4 marks)** *Initial steps:* Download a binary distribution of ArgoUML v0.34 that is suitable for your execution environment (Linux, Windows, or Mac). If you are working on a lab computer in building 303, you should store this distribution either in your echome directory or on your USB pendrive. Unpack the distribution. Search inside the filestructure of the distribution to find argouml.jar. This is an executable jarfile, so you should be able

to execute it (perhaps by double-clicking, if you're on a Windows box) to launch ArgoUML. See http://argouml-stats.tigris.org/documentation/quickguide-0.32/ch02s02.html. Learn how to open zargofiles in ArgoUML (with File/Open Project…), and also learn how to use the File/Export Graphics… command to create a .png graphic image; but do not use umlexamples.zargo during this learning process.

For assessment: Download umlexamples.zargo from the CompSci 230 Assignments webpage, use ArgoUML to create a `.png` graphic image of `Class Diagram 7`, and submit **this image** as your answer to question 6.

7) **(4 marks)** *Initial steps:* Use the "Create/New Class Diagram" command to create a new class diagram called `Company`; then discover how to create a class diagram with the same information as seen in your `.png` image of `Class Diagram 7`. Finally, use the add-method affordance (+) in the `Manager` display, as indicated in the screenshot below, to create an `increaseEmployeePay()` method in the `Manager` class. Note: I have placed this method in the Manager class because I think

- an Employee who is not a Manager should *not* be able to increase anyone's pay, whereas
- a Manager *may* be authorised to increase the pay of some employees.



For assessment: add another method to our `Company` class diagram, after spending a few minutes thinking about this diagram from an OO design perspective. Your submission should include

- **a graphic of your modified `Company` class**, and
- **an English sentence or two** describing why it is appropriate for your newly-added method to be executed by the class in which it is defined.

Please note that there are many possible interpretations of a class diagram, especially when it specifies only the variables and not the methods as in this case. This is not a course in business management, so you will receive full marks for any plausible and understandable description of any reasonably-named method.

Part 3: Huckle Buckle Beanstalk!

8) **(4 marks)** *Initial steps*: Import v1 of the Huckle Buckle Beanstalk codebase (hbbv1.jar) into Eclipse, then compile and run it with no command-line arguments. Note that this jar contains source code – it must be compiled

before it will execute. If your import is successful, a series of lines should be printed to the Eclipse console, starting with

```
        Playing HuckleBuckle on a 5 by 5 grid...
```

Examine the code in hbbv1, to see how it implements each of the elements in the class diagram below.

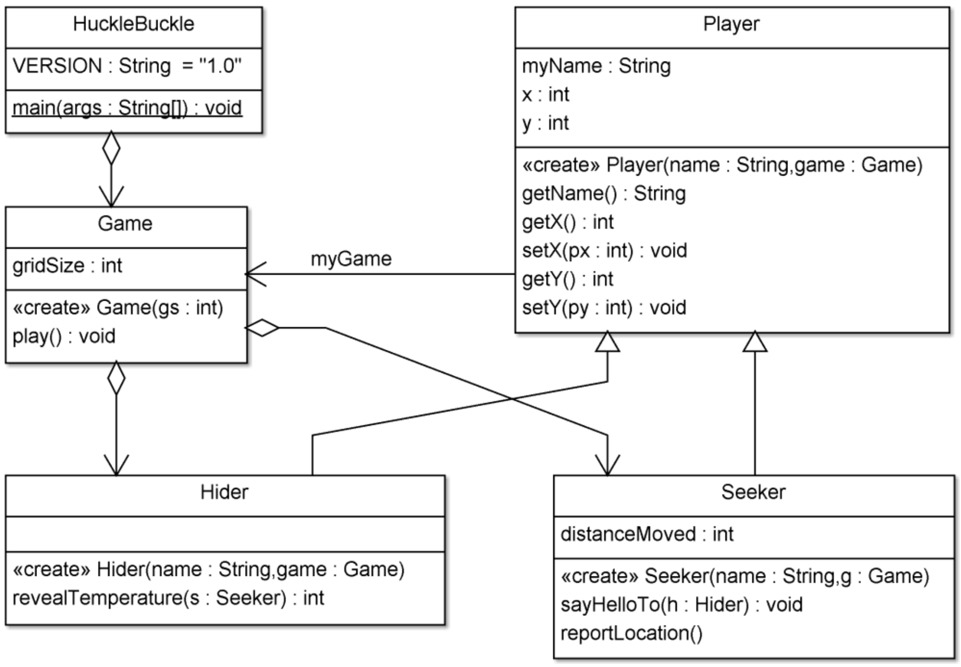


Figure 1. Class diagram for hbbv1.

Now read about "Creating Jar Files" in the Eclipse documentation. Please note that the default jarfile created by Eclipse contains source code, and is not directly executable. It is somewhat more complicated to create what Eclipse calls a "runnable JAR" and which Oracle calls an "executable JAR"; you have seen an example of an executable JAR in Part 2. You are not required to create an executable JAR in this assignment, however you should confirm that hbbv1.jar is *not* runnable on your platform (for example, it won't do anything if you double-click on it).

Now create a new project called hbbv1.1. You'll be using this project to create a new minor version, and it's a good idea to keep a copy of all previous released versions of a code – normally you'd do this in a source code repository but using a repository is out of scope for this assignment.

Import all the source code from hbbv1.jar into your hbbv1.1 project. Note: you could drag-and-drop files from your hbbv1 project using Eclipse's package manager, but you might drop the files in an incorrect place -- so it is safer to use Eclipse's Import functionality whenever you are adding resources (source code, graphic images, etc.) to a project. Change the VERSION number to "1.1". Make sure you can compile and run hbbv1.1 without errors. Note that you will have multiple Run configurations in your Eclipse workspace, so you'll have to be careful to run hbbv1.1 rather than hbbv1, and you should confirm that you're running version 1.1 by looking at the first line of console output.

For assessment:

Add the following paragraphs to the documentation comment for HuckleBuckle.java:

```
     * <p>
     * Second optional command-line parameter: ngames, the number of games to be
     * played on this run of the program, an integer between 0 and 9 inclusive.
     * Note: if only one command-line parameter is supplied, it is interpreted as
     * gridsize rather than as ngames, as indicated by the nested bracketing
     * on the usage comment below.
     * <p>
     * Usage: hbbv1.1 [gridsize [ngames]]
```

Now adjust the code in hbbv1.1 so that it implements the change described above. Note that you won't be able to do this until you understand how the `main()` of v1.0 was testing `args.length` and what is happening when it executes `Integer.parseInt(args[0])`. You may ask anyone for assistance in understanding the code in `hbbv1.0`, but your work on `hbbv1.1` must be independent because it will be assessed.

Your submission on this question should be

- a listing (that is, a human-readable rendition) of **your source code for `main()` of `hbbv1.1`**,
- a listing of **your source code for the method in which you have added a looping construct** (perhaps a for-loop) with `ngames` as a parameter,
- a listing of the **console output produced by a run of `hbbv1.1` with command-line arguments `3 2`**, and
- a listing of the **console output produced by a run of `hbbv1.1` with command-line arguments `4 0`**.

To receive full marks, your console output must tell an understandable story about Sally playing with Harry – so you should insert code to make Sally say additional things (perhaps "Let's play again!" or "No, thanks, not this time.") at appropriate times.

9) **(4 marks)** *Initial steps:* Examine the implementation of `seek()`, and look for repetitive code sequences in this code. You may eventually notice that there are two `if()` statements that have a very similar structure: the seeker's current position (obtained via a getter) is compared to their next position, and if there is any difference then the seeker is moved to this new position (using a setter), the `distanceMoved` variable is updated, and the new position is reported to the console.

For assessment:

Create a new project called hbbv1.2, with a copy of your hbbv1.1 codebase. Add a new method to the `Seeker` class, so that the `seek()` method can be implemented in a simpler fashion, without repetitive `if()` statements. The new method should be called `moveTo()`, and your new implementation of `seek()` should include the following lines of code:

```
for (int y = 0; y < getGame().getGridSize(); y++) {
      for (int cx = 0; cx < getGame().getGridSize(); cx++) {
            // cx: counts the number of times I have moved on row y
            int x = (moveRight ? cx : getGame().getGridSize() - cx - 1);
            moveTo(x,y);
```

Note that a v1.2 `Seeker` can use her `moveTo()` method to move to any position on the grid. Accordingly, your implementation of `moveTo()` should increase `distanceMoved` by the number of steps a seeker must take to get to a new square on the grid; and if the seeker is already at this new position, then she shouldn't report it. You need not check for illegal (out of range) positions on the grid; we'll discuss such issues in the code quality theme of the course.

You'll need a `distance()` function; to keep things simple you might be tempted to copy a private distance() method you'll find in the current implementation of the `Hider`, but to keep the code "simple" (and to receive full marks) you should shift the `Hider's distance()` method into the `Player` class -- so that the `Seeker` can use it too, and so you're not duplicating code.

The process of "cleaning up code" by "pulling a method up to its superclass" (as you have done with `distance()` in this question), by "extracting" lines of code into a new method (as you have done with `moveTo()`), or by any other method, is called re-factoring. Recognising a situation where a re-factoring would be helpful is a very important skill in programming, but it's quite an advanced skill – and I won't be covering the full range of refactoring operations in this course! However you should look at the submenu under Refactor in the Java perspective of Eclipse, to get a sense of the variety of refactoring operations that are important enough – and which are simple enough – to be partly-automated by Eclipse. I do *not* encourage you to use the refactoring support that is available in Eclipse until you are able to refactor without its support, but I hope you can see (after completing this question) that it'd be pretty easy to introduce a bug into a program when you're refactoring it, and that there's a lot of repetitive work to be done when you're refactoring code, so gaining fluency with an IDE's refactoring support is a very important skill for a "production programmer"!

Submit: **a listing of your revised `Seeker` class** (so that your marker can assess your revisions to `seek()` and your implementation of `moveTo()`).

10) **(3 marks)** *Initial steps:* consider the class diagram below, in which I have suggested a major refactoring of the `Game` class for a new version (1.3) of `HuckleBuckle`. I was unsatisfied with the class diagram of earlier versions, because the `Game` in those versions was controlling the human players – they were essentially zombies. I'd prefer to think of a game of `Huckle Buckle` as a set of rules – it should have no methods. Harry and Sally should have some methods which allow them to interact with their environment in a more-or-less natural way. This train of thought led me to the class diagram of Figure 2 below.

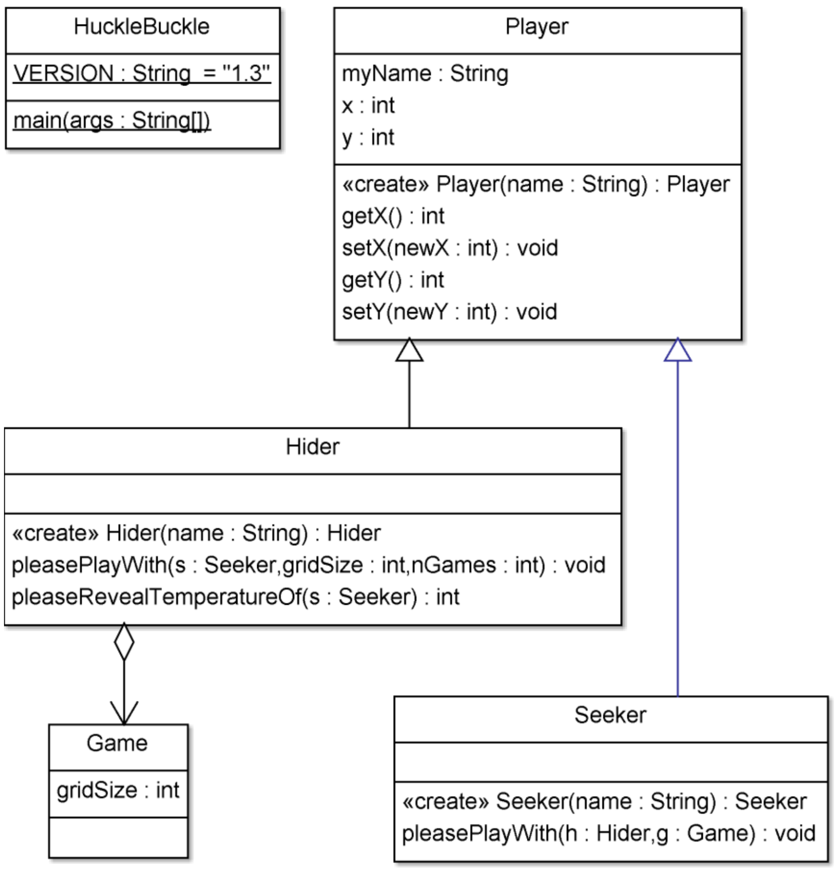

HuckleBuckle
VERSION : String = "1.3"
main(args : String[])
Player
myName : String
x : int
y : int
«create» Player(name : String) : Player
getX() : int
setX(newX : int) : void
getY() : int
setY(newY : int) : void
Hider
«create» Hider(name : String) : Hider
pleasePlayWith(s : Seeker,gridSize : int,nGames : int) : void
pleaseRevealTemperatureOf(s : Seeker) : int
Game
gridSize : int
Seeker
«create» Seeker(name : String) : Seeker
pleasePlayWith(h : Hider,g : Game) : void

Figure 2. Class diagram for V1.3 of HuckleBuckle.

For assessment:

Produce Version 1.3 of the HuckleBuckle codebase. In this version, main() should read its arguments, instantiate a hider named Harry and a seeker named Sally, then invoke Harry's `pleasePlayWith()` method (so that Harry will play zero or more games with Sally). Harry's `pleasePlayWith()` method should be implemented with code which causes him to invoke Sally's `pleasePlayWith()` method – this causes Sally to play a single game of HuckleBuckle. Note that Sally's `pleasePlayWith()` method is essentially the same as her seek() method in earlier versions, but its references to the current game must be adjusted. Sally will eventually find the hidden object, and Harry will "notice" this event when he receives a (void) result from his call to `s.pleasePlayWith(this, new Game(gridSize))`. Harry's `pleasePlayWith()` method should then decrement his `nGames` counter, and (depending on whether or not the counter has reached 0) this method should either exit or ask Sally to play another game (after Harry "hides" another object).

I think you'll find that this OO design for HuckleBuckle doesn't significantly change the amount of code in its implementation. However you might find it somewhat more "elegant" or "natural" to represent players as agents who are interacting with their environment and to represent the game as a class without any instance methods, rather than to represent (as in earlier versions) players as zombies that are controlled by a very powerful Game.

Submit:

- **listings of all classes in `hbbv1.3`**,
- a sample of its **output (for commandline arguments 5 2)**, and
- **a jarfile** containing your source code for `hbbv1.3`.