

# Assignment 1

## Sample Answers and Marking Guide

CompSci 230 S2 2015

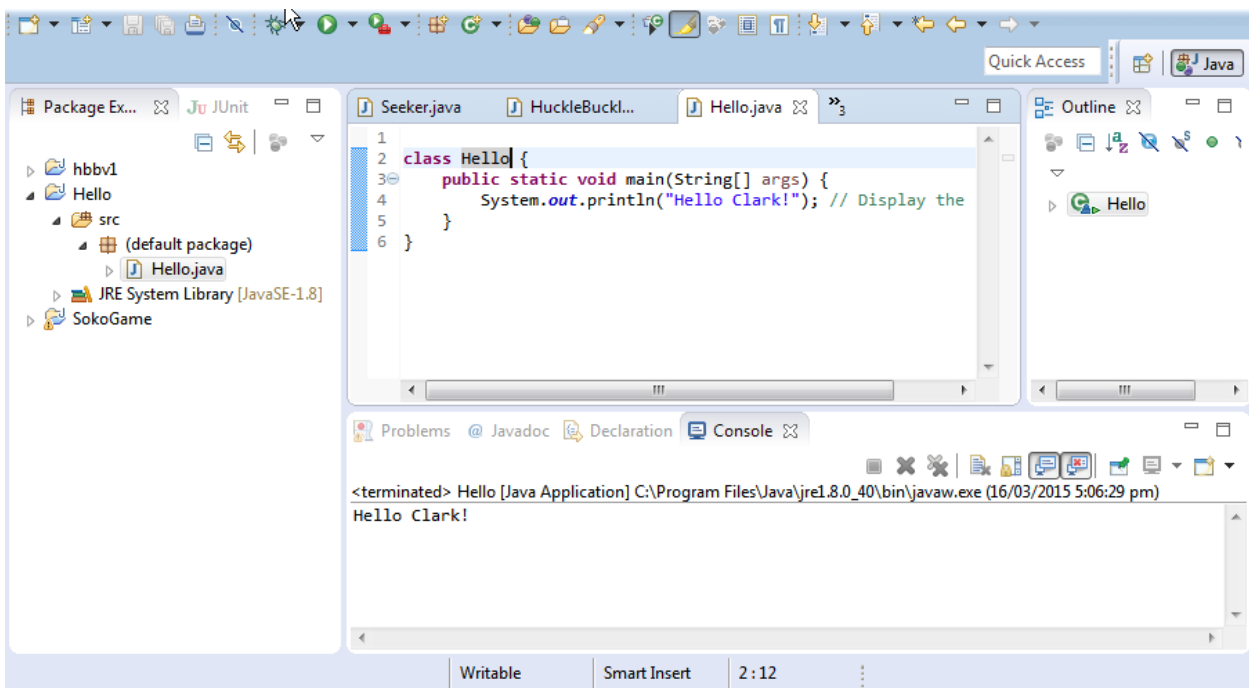
Clark Thomborson

Version 1.0 of 2015-03-30

### 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.
- 1 mark, if the submission contains all of the required info on the first page
- 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.
- 4 marks, if the student's name appears immediately after "Hello" in their console output. Note: students may use any name for their class.

Sample answer:



```
1
2 class Hello {
3     public static void main(String[] args) {
4         System.out.println("Hello Clark!"); // Display the
5     }
6 }
```

<terminated> Hello [Java Application] C:\Program Files\Java\jre1.8.0\_40\bin\javaw.exe (16/03/2015 5:06:29 pm)  
Hello Clark!

- 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.
    - 1 mark, if the comment starts with /\*\*

- 1 mark, if the comment contains a brief (1-sentence) and accurate description of the output of this program (as shown in the screenshot of the previous question).
- 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.
  - 1 mark, if the class contains a main() method which will print "Hello XXX!" to System.out, where XXX is the student's name. Note: the wording of my question implies that the student should be defining the Hello class, but they may use any name for this class.
  - 1 mark, if the class is formatted in a recognisable style for whitespace in Java. Do not mark on non-whitespace formatting elements such as the use of coloured, boldface, or italic fonts.

Sample answer:

```
class Hello {
    /**
     * The Hello class implements an application that
     * simply prints "Hello Clark!" to standard output.
     */
    public static void main(String[] args) {
        System.out.println("Hello Clark!"); // Display the string.
    }
}
```

- 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
```

... 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... 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... click "Run"...

- 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.
- 1 mark, if the output is correct (0.5 marks) and not a screenshot (0.5 marks). Sample answer:

```
abcd
bacd
cbad
bcad
cabd
acbd
```

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:

- the program arguments you selected,
  - 1 mark, for an answer which specifies one set of argument values to be tested
- the output or error message your application produced when given these arguments, and
  - 1 mark, for an answer which reports a plausible output (you need not confirm correctness)
- an English sentence explaining why (or how) you selected these arguments.
  - 1 mark, for an understandable explanation of this student’s test-generation process. Note: students have not received any tuition on unit-testing, so they should receive full marks for a very informal generation process. Do not mark down for grammar or spelling, but please add a summative comment along the following lines:
    - Nicely expressed!
    - Your answer is easily understandable, but it has some minor defects which you could remedy. I’d encourage you to improve your “soft skills” in academic/professional English expression. English-literacy skills are highly valued by local employers, and a high level of proficiency in English expression is required in our advanced degrees (Master of Science, Doctor of Science).
    - Your answer is understandable, but it is poorly expressed. I’m not trained to analyse your writing skills; but I’m pretty sure there are one or more significant defects in this sample of your writing. Our academic staff and our local employers expect all of our graduates to produce syntactically-correct English sentences which are easily understandable. Please consider taking advantage of our University’s ELE centre. See <http://www.library.auckland.ac.nz/services/student-learning/ele>.
    - Sorry, your answer is not understandable. I’d encourage you to put additional effort into improving your skills in written English. See <https://www.auckland.ac.nz/en/for/current-students/cs-academic-information/cs-english-language-support.html> for some ideas.

Sample answer:

I selected “10 abcd” because I was curious to see how this program would behave with a larger value for its first argument (n). I observed the following output:

```
abcd
bacd
cbad
bcad
cabd
acbd
dabc
adbc
badc
abdc
bdac
dbac
cdab
dcab
adcb
dacb
acdb
cadb
bcda
```

```
cbda
dcba
cdba
dbca
bdca
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at Permuter.swap(Permuter.java:15)
    at Permuter.permute(Permuter.java:8)
    at Permuter.permute(Permuter.java:7)
    at Permuter.permute(Permuter.java:7)
    at Permuter.permute(Permuter.java:7)
    at Permuter.permute(Permuter.java:7)
    at Permuter.permute(Permuter.java:7)
    at Permuter.permute(Permuter.java:7)
    at Permuter.main(Permuter.java:20)
```

4) **(4 marks)** 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. ... 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.

- a) Type an input to `TempConv`. In your submission document, report the input you typed and the output you observed.
- 1 mark, for clearly specifying an input that is delimited by quotation marks, a font-change, or some other way (which they must explain at least briefly). The student need not explain their delimiter if they use one of the input-delimitation styles (double-quotation marks, or fixed-width font) that was defined in the assignment document.
  - 1 mark, for including the output produced *before* the student types their input.
  - 1 mark, for including the output produced *after* the student types their input. Note: if the student choose a non-numeric input, this output will include a runtime error message.

Sample answer: I typed "12" (without the quotes). All of the console output I observed, including my echoed input, is listed below:

```
Enter the temperature in F:
12
The temperature in C is: -11.111111111111111
```

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

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

- b) Test the modified `TempConv` application using your input from part a), and report on the output you observe.
- 1 mark, for including output shows either a runtime error message or a floating-point number with one digit after the decimal point.

Sample answer:

```
Enter the temperature in F:
12
The temperature in C is: -11.1
```

5) **(4 marks)** 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 ...

```
/**
 * 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.

- 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),
    - 1 mark, for a JavaDoc comment which describes an understandable new feature that is "not quite trivial" – this phrase should be interpreted broadly, as meaning "any user-perceptible change in program behaviour".
    - 1 mark, for a JavaDoc comment with @version = 1.1. Note that we're not marking the @author tag, but it would be quite appropriate for the student to claim authorship of this code if they retain the reference to java4python.
    - 1 mark, for a listing which contains code implementing the new feature
  - and add a screenshot (or at most two) to illustrate your new feature.
    - 1 mark, for a screenshot illustrating the new feature

Sample answer (with original version commented-out, for convenient reference by markers):

```

import javax.swing.*;

/**
 * GUI which converts a temperature from °F to °C with single-digit output precision
 * <p>
 * Adapted from http://interactivepython.org/courselib/static/java4python
 *
 * @author ctho065
 * @version 1.1
 *
 */
public class TempConvGUI {

    public static void main(String[] args) {
        String fahrString;
        double fahr, cel;

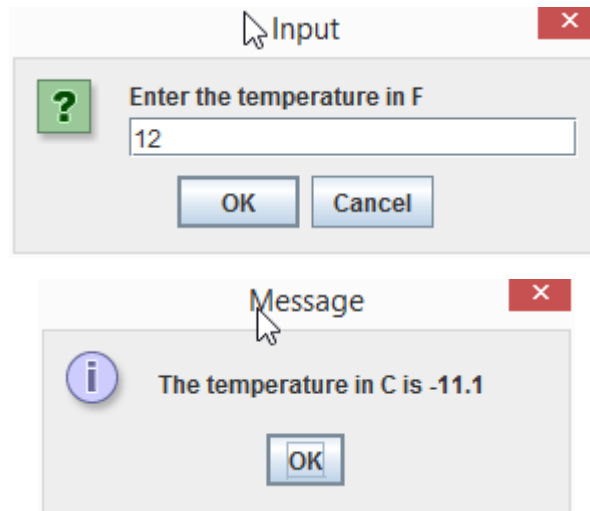
        fahrString = JOptionPane.showInputDialog("Enter the temperature in F");
        fahr = Double.parseDouble(fahrString);
        cel = (fahr - 32) * 5.0/9.0;

        // JOptionPane.showMessageDialog(null,"The temperature in C is, " + cel);

        JOptionPane.showMessageDialog(null,
            "The temperature in C is " + String.format("%.1f", cel));
    }
}

```

}

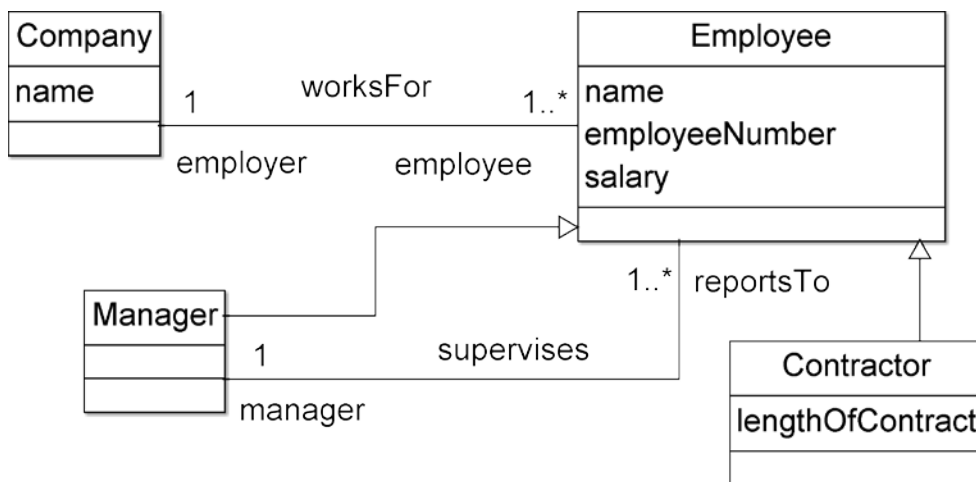


## Part 2: Running an executable jarfile (ArgoUML)

6) (4 marks) Download a [binary distribution of ArgoUML v0.34](#) ... 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 ... 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.

- 4 marks, for any legible rendition of this class diagram as an embedded graphic.

Sample answer:



7) (4 marks) use the add-method affordance (+) in the Manager display... 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

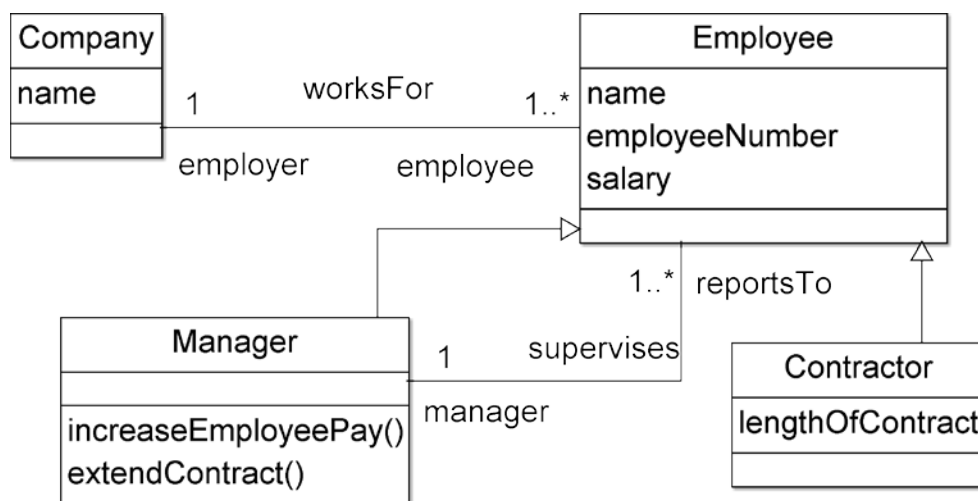
- a graphic of your modified Company class, and

- 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.

- 1 mark, for a class diagram showing an `increaseEmployeePay()` method in the `Manager` class and an additional method in any class
- 1 mark, for understandably specifying (in English) the class to which the new method has been added
- 1 mark, for understandably specifying (in English) the name of the newly-added method
- 1 mark, for understandably indicating a reason why the new method is appropriate to the class in which it is defined

Sample answer: I added an `extendContract()` method to the `Manager` class, because I think only a company manager should be allowed to adjust the terms of a contractor's contract. Please see my class diagram below:



### Part 3: Huckle Buckle Beanstalk!

8) **(4 marks) Assessable work:** 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`
  - 0.5 marks for setting `nGames = 1` by default
  - 0.5 marks for checking the lower (0) and upper (9) limit on `args[1]` input, if provided
- 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
  - 0.5 marks for adding a loop to either `main()` or `play()`
- a listing (that is, a human-readable rendition) of the console output produced by a run of `hbbv1.1` with command-line arguments `3 2`
  - 0.5 marks if the output shows an understandable transition (e.g. Sally saying “Let’s play again!”) between the first and second games,
  - 0.5 marks if Sally starts the second game at (0,0), rather than “jumping” with miscounted steps at the beginning of this game (note that the student isn’t required to submit a listing of their Seeker class – so you’ll have to assign marks solely by examining the output transcript, and if the output shows Sally finding the object at (0,0) in the first game you should award these 0.5 marks because the student’s code *might* have correctly handled the case when Sally has to reset her position and `distanceCounter` at the beginning of the second game),
  - 0.5 marks if Sally does not over-count her steps on the second game (as will happen on most runs of two or more games if her `distanceCounter` isn’t reset), and
- a listing of the console output produced by a run of `hbbv1.1` with command-line arguments `4 0`
  - 1 mark for exiting immediately after printing out both command-line arguments, or for exiting without any output, when `nGames == 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.

Sample answer:

```
package hucklebuckle;

/**
 * A non-interactive console application which simulates a children's game.
 * <p>
 * Optional command-line parameter: gridsize, the size of the playing field, an
 * integer between 1 and 40 inclusive.
 * <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]]
 * <p>
 * When versioning this code, programmers
 * should adjust the value of HuckleBuckle.VERSION so that an accurate version
 * number appears on the first line of program output.
 *
 * @author Clark Thomborson
 */

public class HuckleBuckle {

    public static final String VERSION = "1.1";
```



```

public static void main(String[] args) {

    int gridSize = 5; // gridSize == 5, if there are no command-line args
    int nGames = 1; // play one game, if there isn't a second command-line arg

    if (args.length > 0) {
        try {
            gridSize = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.err
                .println("Error: first arg (gridSize) must be an integer. ");
            System.exit(1);
        }
        if (gridSize < 1 || gridSize > 40) {
            System.err
                .println("Error: first arg (gridSize) must be in the range 1..40. ");
            System.exit(1);
        }
    }
    if (args.length > 1) {
        try {
            nGames = Integer.parseInt(args[1]);
        } catch (NumberFormatException e) {
            System.err
                .println("Error: second arg (nGames) must be an integer. ");
            System.exit(1);
        }
        if (nGames < 0 || nGames > 10) {
            System.err
                .println("Error: second arg (nGames) must be in the range 0..9. ");
            System.exit(1);
        }
    }
    if (args.length > 2) {
        System.err.println("Warning: too many args. ");
    }

    Game myGame = new Game(gridSize);

    myGame.play(nGames);

}

}

void play(int nGames) {
    System.out.println("Playing HuckleBuckle (v" + HuckleBuckle.VERSION
        + ") " + nGames + " times on a " + gridSize + " by " + gridSize
        + " grid...");
    myHider.introduceYourself();
    mySeeker.sayHelloTo(myHider); // let Sally know who will answer her questions
    while (nGames-- > 0) {
        myHider.hide(); // tell Harry to hide an object
        mySeeker.seek(); // tell Sally to start looking
        if (nGames != 0) {
            mySeeker.askToPlayAgain(); // Sally wants to play another game
        }
    }
}
}

```

```

Playing HuckleBuckle (v1.1) 2 times on a 3 by 3 grid...
Harry says "Hi, I'm Harry, let's play Huckle Buckle!"
Sally says, "Hi, Harry, I'm Sally. Glad to meet you!"
Harry says "I'm hiding an object now, I bet you can't find it!"

```

Sally says, "I'm at 0, 0."  
 Harry says to Sally, "You're cool."  
 Sally says, "I'm at 1, 0."  
 Harry says to Sally, "You're cold."  
 Sally says, "I'm at 2, 0."  
 Harry says to Sally, "You're cold."  
 Sally says, "I'm at 2, 1."  
 Harry says to Sally, "You're cold."  
 Sally says, "I'm at 1, 1."  
 Harry says to Sally, "You're warm."  
 Sally says, "I'm at 0, 1."  
 Harry says to Sally, "You're hot."  
 Sally says, "I'm at 0, 2."  
 Harry says to Sally, "Huckle buckle beanstalk!"  
 Sally says, "That was fun! I walked 6 steps before I found it."

Sally says, "Let's play again!"  
 Harry says "I'm hiding an object now, I bet you can't find it!"  
 Sally says, "I'm at 0, 0."  
 Harry says to Sally, "You're cold."  
 Sally says, "I'm at 1, 0."  
 Harry says to Sally, "You're warm."  
 Sally says, "I'm at 2, 0."  
 Harry says to Sally, "You're hot."  
 Sally says, "I'm at 2, 1."  
 Harry says to Sally, "Huckle buckle beanstalk!"  
 Sally says, "That was fun! I walked 3 steps before I found it."

9) **(4 marks) Assessed work:** 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);
  }
}
```

Submit: a listing of your revised *Seeker* class, so that your marker can assess

- your revisions to *seek()*
  - 1 mark, if the nested *for()* loops of *seek()* with the lines of code specified above, and with the "did I find it?" test appearing immediately after these lines of code
- and your implementation of *moveTo()*.
  - 1 mark, if the *distance()* method is not defined in the *Seeker* class (you should assume it is defined in the *Hider* class, as required, but the student wasn't required to list that class so you can't mark on this),
  - 1 mark, for (apparent) correctness, in particular that the *distanceMoved* counter is updated accurately, that *setX(px)* and *setY(py)* are invoked, and that the *reportLocation()* method is invoked only when the distance moved is non-zero;
  - 1 mark, if the *if()* test on *getY()* and the *if()* test on *getX()* in *v1.1 seek()* have been coalesced into a single *if()* test in *moveTo()*. This additional refactoring wasn't explicitly required in the question, but it is a significant simplification -- so I would expect the strongest students in *CompSci 230* to implement *moveTo()* in this way.

- Comment (in a comment) any student who documents the fact that the `moveTo()` method is underspecified in this question, because `distance()` returns a double but `distanceMoved()` is an int. It *may* be considered appropriate (by some stakeholders) to redefine `distanceMoved` so that it is a double; and it *may* be appropriate to truncate or round each of the Seeker's movements before adding it to `distanceMoved`. I have chosen an explicit truncation, in my sample code, by casting with an `(int)`; but the v1.2 `seek()` doesn't move diagonally, so it doesn't matter (in this version!) how the student handles this conversion in their `moveTo()`.

Note: you should **neither** deduct **nor** add marks if there are "unnecessary" invocations of the `setX()`, `setY()`, or `distance()` methods. In my sample code below, I have minimised the complexity of my code by using a single `if()` statement with a minimal number of statements in its block. However some students may assume (or "guess") that adding a local variable and/or introducing an additional `if()` test in `moveTo()` would be an effective way to increase the runtime efficiency of `HuckleBuckle`. At the other extreme, some students may unconditionally execute `setX(px)` and `setY(py)` and the update of `distanceMoved`. Such "unnecessary" executions will not affect the correctness of the code so no marks should be deducted. Runtime efficiency will not be measurably affected by any of these decisions, and anyway runtime efficiency is outside the scope of this course -- until (in Theme D) we're discussing the (very profound) inefficiencies that can be observed in some poorly-designed multithreaded applications.

Sample answer:

```
package hucklebuckle;

class Seeker extends Player {

    private Hider myHider;
    private int distanceMoved; // updated after every change of position

    Seeker(String name, Game game) {
        super(name, game);
        distanceMoved = 0;
    }

    void sayHelloTo(Hider h) {
        myHider = h;
        System.out.println(getName() + " says, \"Hi, \" + h.getName() + ", I'm "
            + getName() + ". Glad to meet you!\");
    }

    private void reportLocation() {
        System.out.println(getName() + " says, \"I'm at \" + getX() + ", "
            + getY() + ".\");
    }

    void askToPlayAgain() {
        // reset distance counter and location: important when multiple games
        // are played!
        distanceMoved = 0;
        setX(0);
        setY(0);
        System.out.println(getName() + " says, \"Let's play again!\");
        System.out.println();
    }

    /**
     * Moves this seeker to (px, py).
     */
}
```

```

* @param px
* the x-coord of the next position of this seeker
* @param py
* the y-coord of the next position of this seeker
*
* If the distance moved is non-zero, the seeker reports their
* new location.
*
* Constraint: this method should use Player.distance()
*/
void moveTo(int px, int py) {
    if (distance(px, py) > 0.0) {
        distanceMoved += (int) distance(px, py);
        // To do: confirm that truncation is appropriate here, e.g.
        // that a Seeker who moves diagonally should count "one step"
        // for distances in the range [1.0, 2.0).
        setX(px);
        setY(py);
        reportLocation();
    }
}

void seek() {
    reportLocation(); // I report my initial location
    boolean moveRight = true; // I move left-to-right on even-numbered rows
    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);
            if (myHider.revealTemperature(this) == 0) { // Did I find it?
                System.out.println(getName()
                    + " says, \"That was fun! I walked "
                    + distanceMoved + " steps before I found it.\");
                return; // I stop when I find the hidden object
            }
        }
        // I move in the opposite direction on each row of the grid
        moveRight = !moveRight;
        // My movement pattern is sometimes called the snake-like row-major
        // ordering. See http://en.wikipedia.org/wiki/Boustrophedon.
    }
    System.out.println(getName() + " says, \"I'm giving up. I took "
        + distanceMoved + " steps before quitting.\");
}

```

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.

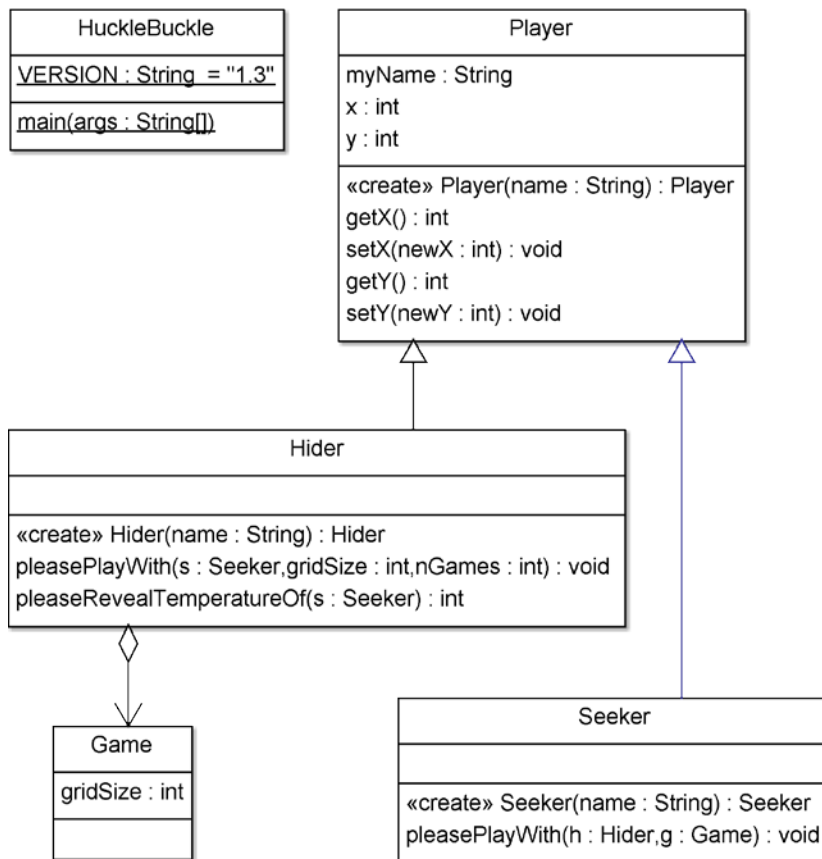


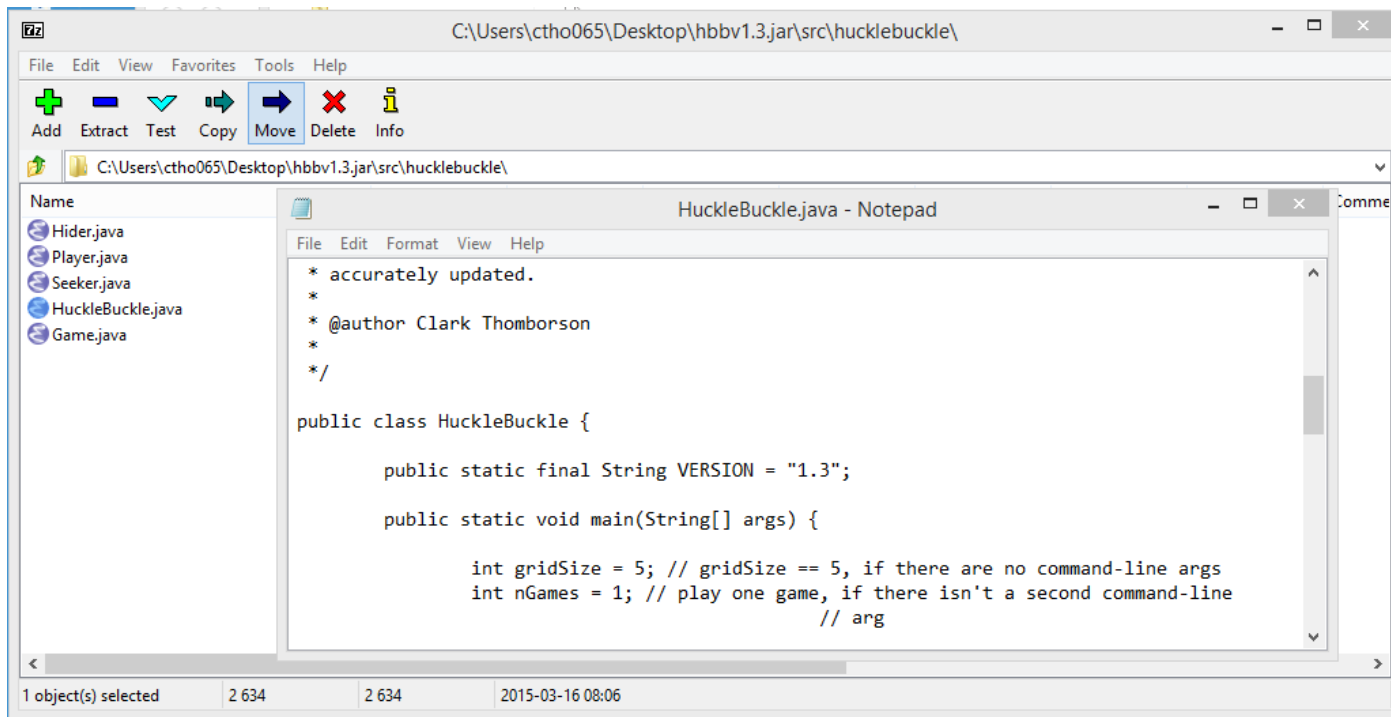
Figure 1. Class diagram for V1.3 of HuckleBuckle.

In Version 1.3, `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).

**Submit:**

- a listing of all classes in `hbbv1.3`,
  - 1 mark, for a `Game` class with no instance methods (aside from getters or setters). Its constructor should initialise its instance variable `gridSize`. Do not mark anything else: this was a very complicated refactoring, and it was incompletely specified. In particular, students should not be penalised for introducing additional instance variables, methods, or parameters. For example, `Player` may be navigable to `Game`, or (as in my sample solution) `Seeker` and `Hider` may both have a `myGame` variable, or (as suggested but not required by Figure 2) there may be no navigability from `Seeker` or `Hider` to `Game` (note that this is possible if a `Game` parameter is added to some methods).
- a sample of its output (for commandline arguments `5 2`),
  - 1 mark, for output that tells an understandable story (0.5 marks) for an apparently-correct (0.5 marks) count of Sally's steps in two games of Huckle Buckle on a 5x5 grid.

- and a jarfile containing your source code for hbbv1.3.
  - 1 mark, for a JARfile that contains a HuckleBuckle.java file for VERSION = "1.3". Note to markers: If you're using a Windows box, you might want to mark this JARfile by opening it with 7zip, navigating to src/HuckleBuckle.java, then by using 7zip's File/Edit function (shortcut F4) to inspect the file as in the screenshot below. With only 60 assignments to mark, I doubt it'd be worthwhile to develop and debug a Windows script using .NET to automate this checking process. However on a Unix box, or using Cygwin on a Windows box, it should take only a few minutes (if you're wizardly about such things ;-)) to develop and debug a script which greps for VERSION in a stream from the uncompressed jar. If you do develop an automated-marking script of this type, please share it with the other markers.



Sample answer:

```
package hucklebuckle;
```

```
/**
 * A non-interactive console application which simulates a children's game.
 * <p>
 * Optional command-line parameter: gridsize, the size of the playing field, an
 * integer between 1 and 40 inclusive.
 * <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: hbbv1.3 [gridsize [ngames]]
 * <p>
 * When versioning this code, programmers should adjust the value of
 * HuckleBuckle.VERSION so that an accurate version number appears on the first
 * line of program output.
 * <p>
 * TODO: put this project into a version-control system, so that all classes
 * (including this one) are version-stamped, so that versions don't get
 * confused, so that versions can be conveniently stored and retrieved and
 * archived, and so that the VERSION in this class is automatically and
 * accurately updated.
```

```

*
* @author Clark Thomborson
*
*/

public class HuckleBuckle {

    public static final String VERSION = "1.3";

    public static void main(String[] args) {

        int gridSize = 5; // gridSize == 5, if there are no command-line args
        int nGames = 1; // play one game, if there isn't a second command-line
            // arg

        if (args.length > 0) {
            try {
                gridSize = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.err
                    .println("Error: first arg (gridSize) must be an integer. ");
                System.exit(1);
            }
            if (gridSize < 1 || gridSize > 40) {
                System.err
                    .println("Error: first arg (gridSize) must be in the range 1..40. ");
                System.exit(1);
            }
        }
        if (args.length > 1) {
            try {
                nGames = Integer.parseInt(args[1]);
            } catch (NumberFormatException e) {
                System.err
                    .println("Error: second arg (nGames) must be an integer. ");
                System.exit(1);
            }
            if (nGames < 0 || nGames > 10) {
                System.err
                    .println("Error: second arg (nGames) must be in the range 0..9. ");
                System.exit(1);
            }
        }
        if (args.length > 2) {
            System.err.println("Warning: too many args. ");
        }

        System.out.println("Playing HuckleBuckle (v" + HuckleBuckle.VERSION
            + ") " + nGames + " times on a " + gridSize + " by " + gridSize
            + " grid...");

        Hider myHider = new Hider("Harry");

        myHider.pleasePlayWith(new Seeker("Sally"), gridSize, nGames);

    }

}

package hucklebuckle;

class Game {

```

```

private int gridSize;

Game(int gs) {
    gridSize = gs;
}

int getGridSize() {
    return gridSize;
}
}

package hucklebuckle;

class Player {

    private String myName;
    private int x; // constraint: 0 <= x < gridSize
    private int y; // constraint: 0 <= y < gridSize
    // TODO: push myGame up from Hider and Seeker, to simplify code
    // and to allow constraint-checking on x and y.

    Player(String name) {
        myName = name;
        setX(0); // all players start at (0,0)
        setY(0);
    }

    String getName() {
        return myName;
    }

    int getX() {
        return x;
    }

    void setX(int px) {
        x = (int) px; // TODO: add bounds-checking logic
    }

    int getY() {
        return y;
    }

    void setY(int py) {
        y = (int) py; // TODO: add bounds-checking logic
    }

    /**
     * Returns the distance of this player to another player
     */
    double distance(Player other) {
        return distance(other.getX(), other.getY());
    }

    /**
     * Returns the Euclidean distance from this player to (px, py)
     *
     * @param px
     *         x-coord of another point on a 2d grid
     * @param py
     *         y-coord of another point on a 2d grid
     * @return a double >= 0.0
     */
}

```



```

    double distance(int px, int py) {
        return Math.hypot(getX() - px, getY() - py);
    }
}

package hucklebuckle;

class Seeker extends Player {

    private Hider myHider;
    private Game myGame;
    private int nGamesPlayed;
    private int distanceMoved; // updated after every change of position

    Seeker(String name) {
        super(name);
        nGamesPlayed = 0;
    }

    void pleasePlayWith(Hider h, Game g) {
        myHider = h;
        myGame = g;
        if(nGamesPlayed == 0){
            System.out.println(getName() + " says, \"Hi, \" + myHider.getName() + ", I'm "
                + getName() + ". Glad to meet you! Are you ready?\"");
        } else {
            System.out.println(getName() + " says, \"OK! Are you ready?\"");
        }
        myHider.pleaseHideSomething();
        seek();
        nGamesPlayed++;
    }

    private void reportLocation() {
        System.out.println(getName() + " says, \"I'm at \" + getX() + ", \"
            + getY() + ".\");
    }

    /**
     * Moves this seeker to (px, py).
     *
     * @param px
     * the x-coord of the next position of this seeker
     * @param py
     * the y-coord of the next position of this seeker
     *
     * If the distance moved is non-zero, the seeker reports their
     * new location.
     *
     * Constraint: this method should use Player.distance()
     */
    private void moveTo(int px, int py) {
        if (distance(px, py) > 0.0) {
            distanceMoved += (int) distance(px, py);
            // To do: confirm that truncation is appropriate here, e.g.
            // that a Seeker who moves diagonally should count "one step"
            // for distances in the range [1.0, 2.0).
            setX(px);
            setY(py);
            reportLocation();
        }
    }
}

```

```

}

private void seek() {
    distanceMoved = 0;
    setX(0);
    setY(0);
    reportLocation(); // I report my initial location
    boolean moveRight = true; // I move left-to-right on even-numbered rows
    for (int y = 0; y < myGame.getGridSize(); y++) {
        for (int cx = 0; cx < myGame.getGridSize(); cx++) {
            // cx: counts the number of times I have moved on row y
            int x = (moveRight ? cx : myGame.getGridSize() - cx - 1);
            moveTo(x, y);
            if (myHider.pleaseRevealTemperatureOf(this) == 0) { // Did I find it?
                System.out.println(getName()
                    + " says, \"That was fun! I walked "
                    + distanceMoved + " steps before I found it.\");
                return; // I stop when I find the hidden object
            }
        }
        // I move in the opposite direction on each row of the grid
        moveRight = !moveRight;
        // My movement pattern is sometimes called the snake-like row-major
        // ordering. See http://en.wikipedia.org/wiki/Boustrophedon.
    }
    System.out.println(getName() + " says, \"I'm giving up. I took "
        + distanceMoved + " steps before quitting.\");
}
}
}

```

Playing HuckleBuckle (v1.3) 2 times on a 5 by 5 grid...

```

Harry says "Hi, I'm Harry, let's play Huckle Buckle!"
Sally says, "Hi, Harry, I'm Sally. Glad to meet you! Are you ready?"
Harry says "I'm ready now. I bet you can't find it!"
Sally says, "I'm at 0, 0."
Harry says to Sally, "You're cold."
Sally says, "I'm at 1, 0."
Harry says to Sally, "You're cool."
Sally says, "I'm at 2, 0."
Harry says to Sally, "You're cool."
Sally says, "I'm at 3, 0."
Harry says to Sally, "You're cool."
Sally says, "I'm at 4, 0."
Harry says to Sally, "You're cold."
Sally says, "I'm at 4, 1."
Harry says to Sally, "You're cool."
Sally says, "I'm at 3, 1."
Harry says to Sally, "You're warm."
Sally says, "I'm at 2, 1."
Harry says to Sally, "You're warm."
Sally says, "I'm at 1, 1."
Harry says to Sally, "You're warm."
Sally says, "I'm at 0, 1."
Harry says to Sally, "You're cool."
Sally says, "I'm at 0, 2."
Harry says to Sally, "You're warm."
Sally says, "I'm at 1, 2."
Harry says to Sally, "You're hot."
Sally says, "I'm at 2, 2."
Harry says to Sally, "You're boiling!"
Sally says, "I'm at 3, 2."
Harry says to Sally, "You're hot."

```

Sally says, "I'm at 4, 2."  
Harry says to Sally, "You're warm."  
Sally says, "I'm at 4, 3."  
Harry says to Sally, "You're warm."  
Sally says, "I'm at 3, 3."  
Harry says to Sally, "You're boiling!"  
Sally says, "I'm at 2, 3."  
Harry says to Sally, "Huckle buckle beanstalk!"  
Sally says, "That was fun! I walked 17 steps before I found it."

Harry says "Will you play again with me?"  
Sally says, "OK! Are you ready?"  
Harry says "I'm ready now. I bet you can't find it!"  
Sally says, "I'm at 0, 0."  
Harry says to Sally, "You're boiling!"  
Sally says, "I'm at 1, 0."  
Harry says to Sally, "You're hot."  
Sally says, "I'm at 2, 0."  
Harry says to Sally, "You're warm."  
Sally says, "I'm at 3, 0."  
Harry says to Sally, "You're cool."  
Sally says, "I'm at 4, 0."  
Harry says to Sally, "You're cold."  
Sally says, "I'm at 4, 1."  
Harry says to Sally, "You're cold."  
Sally says, "I'm at 3, 1."  
Harry says to Sally, "You're cool."  
Sally says, "I'm at 2, 1."  
Harry says to Sally, "You're warm."  
Sally says, "I'm at 1, 1."  
Harry says to Sally, "You're boiling!"  
Sally says, "I'm at 0, 1."  
Harry says to Sally, "Huckle buckle beanstalk!"  
Sally says, "That was fun! I walked 9 steps before I found it."

Please see hbbv1.3.jar (submitted separately).