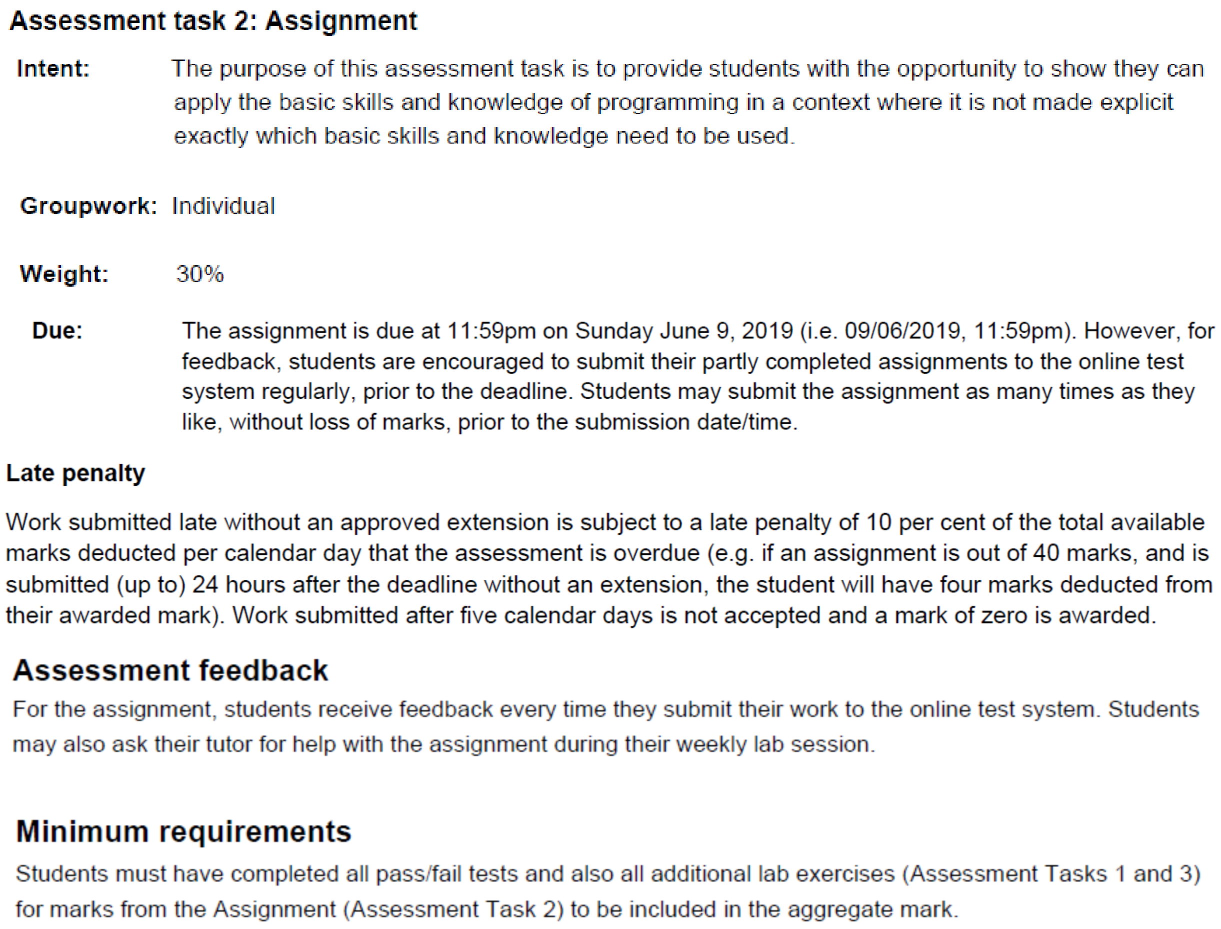
Programming Fundamentals (48023) - Assignment

This assignment will **NOT** be distributed in hardcopy at lectures (or anywhere else).

The following are extracts from the subject outline that are relevant to the assignment …



Assessment task 2: Assignment

Intent: The purpose of this assessment task is to provide students with the opportunity to show they can apply the basic skills and knowledge of programming in a context where it is not made explicit exactly which basic skills and knowledge need to be used.

Groupwork: Individual

Weight: 30%

Due: The assignment is due at 11:59pm on Sunday June 9, 2019 (i.e. 09/06/2019, 11:59pm). However, for feedback, students are encouraged to submit their partly completed assignments to the online test system regularly, prior to the deadline. Students may submit the assignment as many times as they like, without loss of marks, prior to the submission date/time.

Late penalty

Work submitted late without an approved extension is subject to a late penalty of 10 per cent of the total available marks deducted per calendar day that the assessment is overdue (e.g. if an assignment is out of 40 marks, and is submitted (up to) 24 hours after the deadline without an extension, the student will have four marks deducted from their awarded mark). Work submitted after five calendar days is not accepted and a mark of zero is awarded.

Assessment feedback

For the assignment, students receive feedback every time they submit their work to the online test system. Students may also ask their tutor for help with the assignment during their weekly lab session.

Minimum requirements

Students must have completed all pass/fail tests and also all additional lab exercises (Assessment Tasks 1 and 3) for marks from the Assignment (Assessment Task 2) to be included in the aggregate mark.

Exemption to Minimum Requirements: To have their mark counted from Part A of the assignment, students do **NOT** have to complete the last additional lab test on EACH thread (i.e. the lab tests ListOfNV3PartA and ListOfNV3PartA). But students must complete BOTH those two lab tests to have marks counted for Part B of the assignment. This exception allows students to commence working on Part A of the assignment before they have completed those last two lab tests, confident that their mark for Part A will count.

There are two parts to the assignment and both parts have the same due date. You do NOT have to do all parts to register a mark for the assignment. You can stop doing this assessment item at any time, and whatever marks you have in PLATE at that time will be counted.

STUDENTS ARE REMINDED THAT THIS ASSIGNMENT HAS **NO INFLUENCE** ON WHETHER THEY PASS OR FAILTHIS SUBJECT. PASSING AND FAILING IS **SOLELY** DETERMINED BY THE MASTERY TESTS.

Assignment Minimum Requirements

To receive any marks, your solution must meet the following minimum requirements:

1. You must complete this assessment task in the order of the the parts, A and B. You have to use your solution to Part A to do Part B, so you have no option but to do Part A before Part B. But you do NOT need to score full marks on Part A before doing Part B. You will need the full, correct functionality of Part A to do Part B, but you do not need any of the “design” and “indentation” marks to start on Part B. (The exact breakdown of marks for Part A is given later in this document.)
2. Within each of Parts A and B, the tasks must be implemented in the order specified in the “Task “sections below.
3. You may only use the features of Java that are taught in this subject. For example, you must not use inheritance, exceptions, varargs (e.g. printf), interfaces, or generics. We want to assess your ability to use the very specific features of Java that we have taught you in this subject.
4. Your solutions for Parts A and B must NOT use arrays (or equivalent).
5. Your program's output must EXACTLY match the output given by PLATE. To ensure you meet this requirement, it is highly recommended that you submit to PLATE frequently; at least once on each day that you do any work on the assignment.

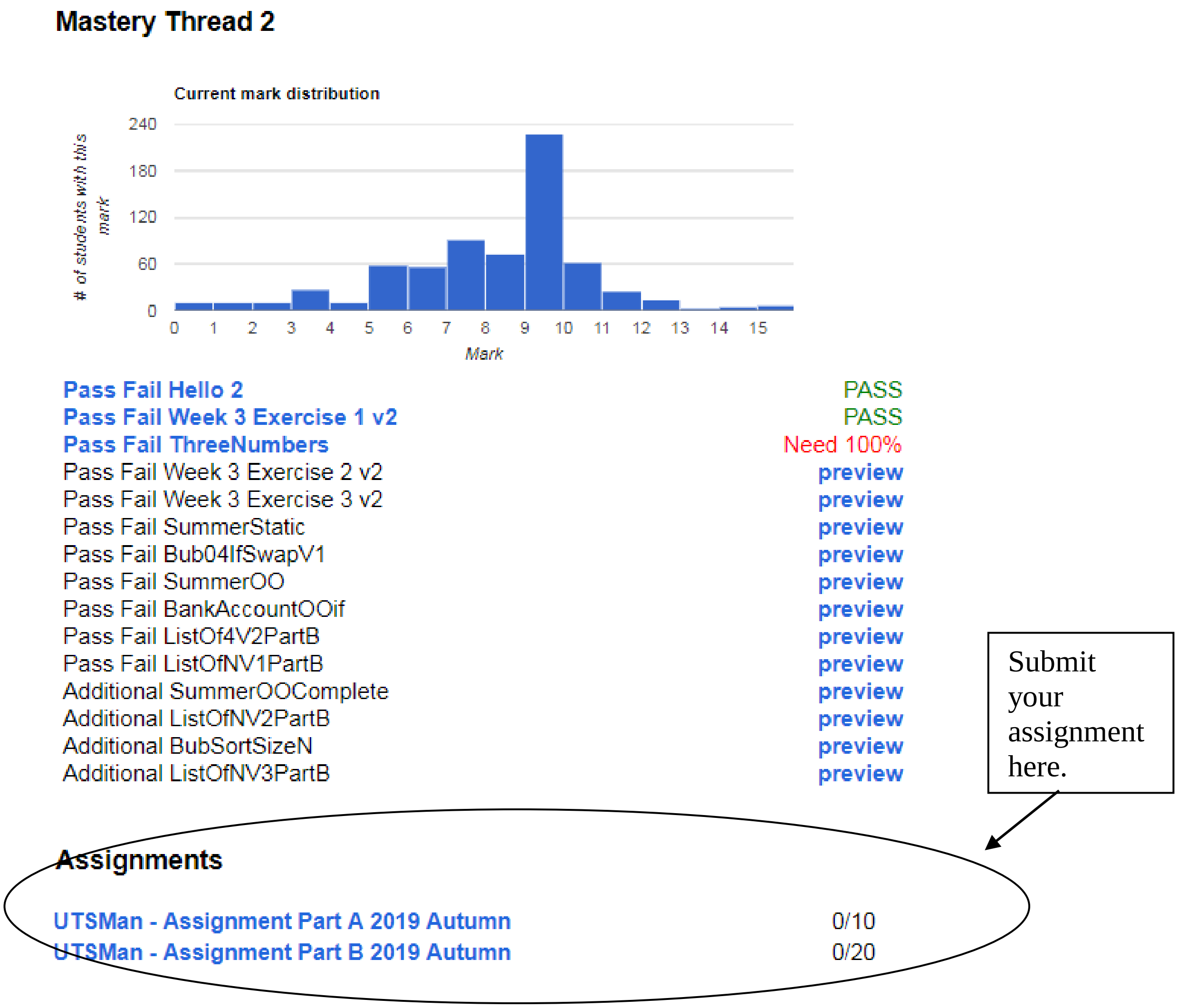
Assignment submission and return

Your assignment must be submitted as a JAR file through the PLATE system, online, at http://plate.it.uts.edu.au/. As shown in the diagram below, you submit your assignment on the same web page where you submit your lab tests; just lower down on that web page.

You may submit and resubmit as many times as you wish, before the due date, and PLATE will update your mark. Your mark for each part is available as soon as you submit to PLATE.

No part of this assignment is manually marked.

Further instructions for submitting to PLATE are displayed online at the PLATE website.



Mastery Thread 2

Current mark distribution

Pass Fail Hello 2	PASS
Pass Fail Week 3 Exercise 1 v2	PASS
Pass Fail ThreeNumbers	Need 100%
Pass Fail Week 3 Exercise 2 v2	preview
Pass Fail Week 3 Exercise 3 v2	preview
Pass Fail SummerStatic	preview
Pass Fail Bub04IfSwapV1	preview
Pass Fail SummerOO	preview
Pass Fail BankAccountOOif	preview
Pass Fail ListOf4V2PartB	preview
Pass Fail ListOfNV1PartB	preview
Additional SummerOOComplete	preview
Additional ListOfNV2PartB	preview
Additional BubSortSizeN	preview
Additional ListOfNV3PartB	preview

Submit your assignment here.

Assignments

UTSMan - Assignment Part A 2019 Autumn	0/10
UTSMan - Assignment Part B 2019 Autumn	0/20

<u>NOTE</u>: Unlike the lab tests, there is no “skeleton” file to download for the assignment.

Special Consideration

If your performance in an assessment item or items has been affected by extenuating or special circumstances beyond your control you may apply for Special Consideration. Information on how to apply can be found at **http://www.sau.uts.edu.au/assessment/consideration.html**

Model solution

A model solution can be seen by emailing the subject coordinator **on or after Sunday July 21 (i.e. 6 weeks after the due date).** This lengthy delay after the due date is because some students are likely to have been granted an extension for sickness, or misadventure (and such illness/misadventure has been documented). Do not email the subject coordinator to request the model solution before Sunday July 21, as he will not be maintaining a list of names and thus your premature request may not be answered. **Model solutions for each part of the assignment are only available to students who submitted that part of the assignment.**

Academic misconduct (and submitting regularly)

This assignment **must be done by yourself** and not with anyone else. **Group work is not allowed** and will be considered as academic misconduct.

Do not show other people your code, or look at other people's code, or discuss assignment code with other people; it is an offence to have a copy of someone else's assignment (before the submission date). **Do not post your assignment on the web.** Posting your assignment on the web and getting help through blogs, forums or other websites is considered to be an academic misconduct.

To detect student misconduct, the subject uses an online system called PLATE available at http://plate.it.uts.edu.au

You should submit your progress to PLATE regularly while you are working on each task. This will provide us with a record that you have been doing your own work. If two students submit the same solution, your submission history may be used by the University Student Conduct Committee to determine who did the work and who copied.

This assignment is divided into separate tasks described below. You must submit your progress to plate regularly while attempting each individual task. That means **a student cannot submit one complete working solution at the end without any prior submissions to PLATE**. This will provide us with a record that you have been doing your own work.

If two students submit the same solution, your submission record may be used by the University Student Conduct Committee to determine which student did the work and which student copied. For more details on assignment submission, return and other important rules, scroll down to the last few pages.

Students may find it useful to consult The UTS Coursework Assessment Policy & Procedure Manual, at http://www.gsu.uts.edu.au/policies/assessment-coursework.html

Expected work load

It is estimated that the workload for Parts A and B are about 4 to 10 hours of work. But some people may complete the task in 5 hours, and some may need 30 hours or more; there is a huge variation in students' experience and abilities.

For Part A, a well-designed solution is expected to use approximately 100 lines of code, excluding comments. For Part B, a well-designed solution is expected to use approximately 300 lines of code, excluding comments.

Seeking Help

Students should make the most of the many opportunities for face-to-face help in labs. Students are welcome to go to a **non-exam** hour of **ANY** lab session to seek help from a tutor. You do NOT have to be enrolled in a lab to get help from a tutor in the non-exam hour. **All tutors should be able to answer most questions about Part A of the assignment.** Some but not all tutors will be able to answer questions about Part B, given a little time to think about your question and the assignment.

Tutor Ryan Heise is be able to answer most questions about all parts of the assignment, having been the original author of Parts A and B. Ryan’s labs are:

- Thu13_08_409_Ryan … Thursday, starting at 1pm, in room CB11.08.409
- Thu15_B1_103_Ryan …Thursday, starting at 3pm, in room CB11.B1.103

The subject coordinator may answer some simple questions by email that require a very short answer. However, if an emailed question would require a lengthy email reply, a student will be told to seek help face-to-face, from a tutor at a lab session or from the lecturer at a lecture session..

Students should NOT request help from tutors via email. Tutors are only paid for their time in the lab. If you email them, you are asking them to work for free ... do you work for free?

Should You Attempt the Assignment?

Students are reminded that they do NOT have to do the assignment to pass this subject. In fact, as the assignment is worth 30% of the subject, a student can score a credit in this subject (i.e. a mark of 65 or higher) without doing the assignment.

However, students who expect to follow this subject by doing Applications Programming (48024) are STRONGLY ENCOURAGED to do Part A of the assignment to prepare for Applications Programming. Of the students who did Programming Fundamentals in Autumn 2017 and who then went on to do Applications Programming (48024) in Spring 2017, **28%** failed Applications Programming. Of those students:

- Of the 33 students who scored **50/P** in Programming Fundamentals, **55%** failed Applications Programming.

- Of the 12 students who scored between **51 and 64 (i.e. a Pass)** in Programming Fundamentals, **33%** failed Applications Programming.
- Of the 20 students who scored **65 (i.e. the minimum mark for a Credit)** in Programming Fundamentals, **35%** failed Applications Programming.
- Of the 22 students who scored between **66 and 74 (i.e. a Credit)** in Programming Fundamentals, **27%** failed Applications Programming.
- Of the 24 students who scored between **75 and 84 (i.e. a Distinction)** in Programming Fundamentals, **8%** failed Applications Programming.
- Of the 34 students who scored between **85 and 100 (i.e. a High Distinction)** in Programming Fundamentals, **12%** failed Applications Programming.

Solution requirements

To receive any marks, your solution must meet the following minimum requirements:

- The tasks must be implemented in the order specified in section "Tasks" below.
- As a general rule, your solution must use only the features of Java that are taught in this subject. For example, students must not use inheritance, exceptions, varargs (e.g. printf), interfaces, or generics. Also, students must not use arrays (or equivalent, such as collections) in Parts A and B, even though arrays are taught in this subject.
- Your program's output must **exactly** match the output given by PLATE. White space (i.e. spaces, tabs and new lines) is significant in PLATE.
- You must define methods with the exact names and parameters as specified below (i.e. you must define methods with the given "*signatures"*), however you are permitted to define any number of additional methods of your own. Unless otherwise stated, you must not add additional fields to classes.

Overview of Parts A and B

In this assignment you will create a simple Pacman-like game with one player, 3 dots, an exit and an enemy. The player can move left, right, up or down and must collect all 3 dots and reach the exit without being killed by the enemy. The enemy is programmed to chase down the player. The game finishes when either:

(1) all the dots have been collected or the exit has been reached, or

(2) when the enemy has killed the player.

Sample output is shown below. Your program should produce input/output in exactly the following format, with user's input shown underlined and in **bold**:

```
Initial x: 5
Initial y: 0
Player[](5,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,5)
Move (l/r/u/d): l
Player[](4,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,4)
Move (l/r/u/d): l
Player[](3,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,3)
Move (l/r/u/d): l
Player[](2,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,2)
Move (l/r/u/d): l
Player[](1,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,1)
Move (l/r/u/d): d
Player[*](1,1) Dot(-,-) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(4,1)
Move (l/r/u/d): r
Player[*](2,1) Dot(-,-) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(3,1)
Move (l/r/u/d): d
Player[**](2,2) Dot(-,-) Dot(-,-) Dot(3,3) Exit(4,4) Enemy(2,1)
Move (l/r/u/d): r
Player[**](3,2) Dot(-,-) Dot(-,-) Dot(3,3) Exit(4,4) Enemy(2,1)
Move (l/r/u/d): d
Player[***](3,3) Dot(-,-) Dot(-,-) Dot(-,-) Exit(4,4) Enemy(2,2)
Move (l/r/u/d): r
Player[***](4,3) Dot(-,-) Dot(-,-) Dot(-,-) Exit(4,4) Enemy(2,3)
Move (l/r/u/d): d
Player[***](4,4) Dot(-,-) Dot(-,-) Dot(-,-) Exit(-,-) Enemy(2,4)
You win!
```

Explanation of the above sample output

The program begins by asking the user to input the initial position for the player. The x coordinate increases from left to right and the **y-coordinate increases** **from top to bottom**. The position of each Player, each Dot, the Exit and the Enemy are shown in the format (x,y). Once a dot has been collected, or the exit has been successfully reached or the player has been killed, its position is instead shown as (-,-) with coordinates replaced by the - symbol. The user plays the game by entering move commands (l/r/u/d) indicating move left, right up or down, until the game is over.

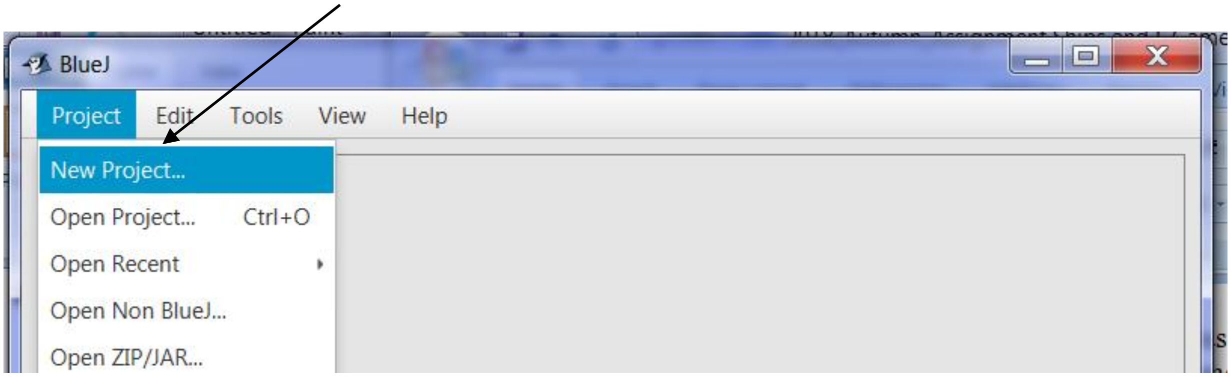
In Part A, you will build only a fragment of this game. In Part B, you will complete all functionality.

Part A

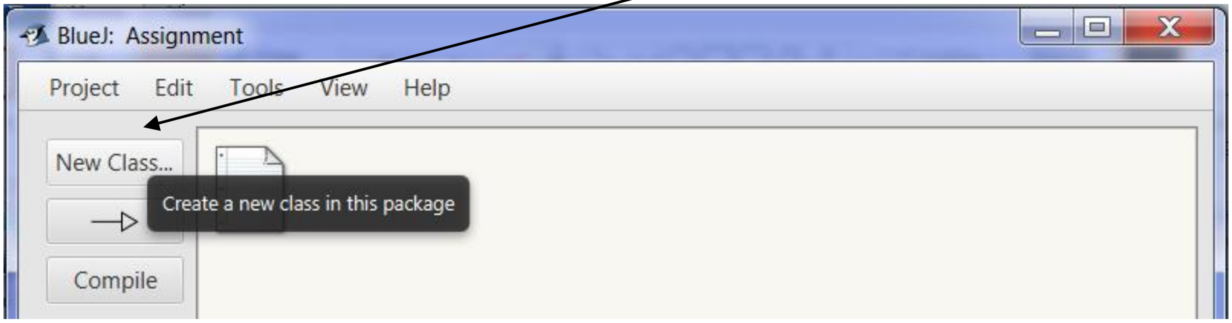
Due date: 11:59pm on Sunday June 9, 2019 (i.e. 9/6/2019, 11:59pm).
Value: 10%
Topics: Data flow, OO programming basics, Control flow.
Objectives: This assignment supports objectives 1-4.

Students who expect to follow this subject by doing Applications Programming (48024) are STRONGLY ENCOURAGED to do Part A of the assignment to prepare for Applications Programming.

NOTE: Unlike the lab tests, there is no "skeleton" file for the assignment. To start the assignment, select "New project" off the "Project" menu, as shown below:



After creating the project, to create each new class, select "New Class" as shown below:



Do NOT save your JAR file to your BlueJ Project Folder

Do NOT save your JAR file into the same folder as your BlueJ files for the assignment. When you do that, each time you make a new JAR file, the new JAR contains the old JAR, and eventually you end up with a REALLY REALLY BIG JAR file. The JAR file for benchmark solution used in PLATE is only 5KB. Your JAR file should not be much bigger than that.

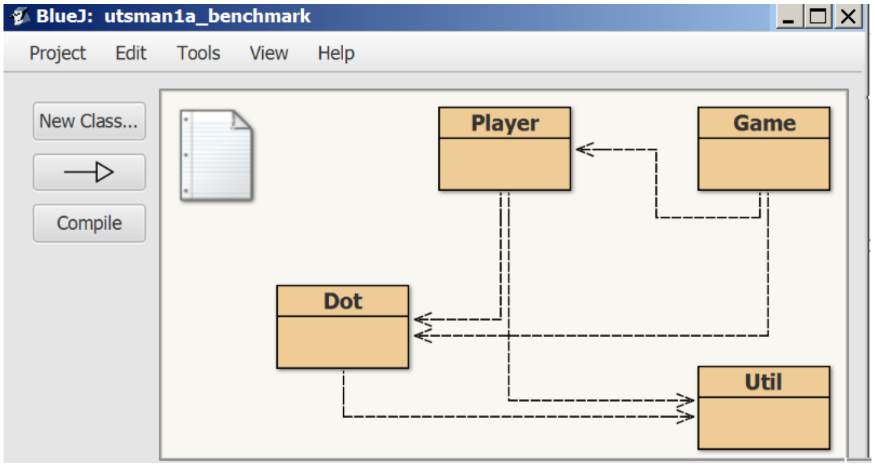
If you have already been saving the assignment JAR file into the same folder as the BlueJ files, then you'll need to:

(a) create a new BlueJ project folder,
(b) copy-and-paste your code from the old folder into the new folder, and then
(c) in future, don't save the jar file into this same new folder as the BlueJ files.

NOTE! THE ASSIGNMENT SUBMISSION SYSTEM RETAINS YOUR MOST RECENT ASSIGNMENT MARK, NOT YOUR HIGHEST ASSIGNMENT MARK. It is therefore possible to submit after the deadline and see your mark for the assignment go **DOWN**.

The coordinator will **NOT** be entertaining people who email to say that they "accidently" re-submtted the assignment after the deadline, or people who say "*I resubmitted just to find out what would happen if* ...". If you submit AFTER the deadline, and your mark goes down, then that lower mark will remain your mark for the assignment.

In this part, you will create the various classes that comprise the game. After completing part A, your project in BlueJ will look something like this:



The actual position of your class icons will probably vary from what is shown above.

Tasks

These tasks must be completed in the order listed below.

Task 1: Classes and fields

In a new BlueJ project write code to produce the following three *classes* (written in italics) and their “fields” / “private data members” / “variables” (enclosed in double quotes):

- A *Game* has 1 instance of the *Player* class, called “player”, and 3 instances of the *Dot* class (called “dot1”, “dot2” and “dot3”). (An instance of a class is an object.)
- A *Player* has an “x” and “y” value (both integers) which together describe the position of the player. The coordinate system of the x and y position is as follows: the x-coordinate increases from left to right, and the **y-coordinate increases <u>from top to</u>**

bottom. The player also keeps a counter (an integer) of the number of dots collected (called “collectedDots”).

- A *Dot* has an “x” and “y” position with the same coordinate system as the player, and also has a boolean status (i.e. a variable which has either the value true or the value false; look it up) called “exists” which is initially true and is set to false whenever the dot has been collected by the player.

You should also add a fourth class, called *Util*, which for Part A is given to you and is as follows:

```
public class Util {
   public static String coordStr(int x, boolean shown) {
        return shown ? (""+x) : "-";
   }

   public static String objectStr(int x, int y, boolean shown) {
        return "(" + coordStr(x, shown) + "," + coordStr(y, shown) + ")";
   }
}
```

Note: *Don’t copy and paste the above code from this PDF document. Instead, copy it from the file “ClassUtilCode.txt” which was attached to the same email that distributed this PDF document. If you have deleted that email without keeping “ClassUtilCode.txt”, it can be found in UTSonline, , under the "Assessment" menu item on the left hand side of UTSonline..*

The “?” operator in the *Util* method called “`coordStr`” has not been discussed in lectures. You don’t need to understand the “?” operator to use *Util*, but if you are curious you can, for example google the words … java tutorial question mark operator.

The names of your classes and fields should exactly match those in the above system description, and must follow Java’s naming conventions (i.e. class names begin with an uppercase letter, field names begin with a lowercase letter, compound nouns (google it) are joined without a space and the second word begins with an uppercase letter).

All classes must be public and all fields must be private.

Submit your project to PLATE now to receive your marks and feedback. You can resubmit unlimited times before the due date. Use the feedback from plate to improve your code and then resubmit to improve your mark.

Task 2: Constructors

In this task, you will add constructors to each class to initialise the program.

2.1 Dot

Define a constructor for class Dot that takes the initial x and y values as parameters, and copies these initial values into the x and y fields. The Dot constructor should also initialise the "exists" field to true.

2.2 Player

Define a constructor for class Player that takes the initial x and y values as parameters, and copies these initial values into the x and y fields. The collectedDots fields should also be initialised to 0.

Submit your code to PLATE to receive marks for this part and do the same for each subsequent part below. You are required to submit your progress to plate while you are making progress.

2.3 Game

Define a constructor for class Game that takes the initial x and y positions of the player as parameters and creates and initialises the player, dot1, dot2 and dot3 fields. *Hint*: creating and initialising player, dot1, dot2 and dot3 is analogous to creating and initialising:

- "scanner" in the lab test class "ThreeNumbers"
- "summer1" or "summer2" in the main method of the lab test class "SummerOO"

The player should be created at the position given by the parameters. dot1, dot2 and dot3 must be created at positions (1,1), (2,2) and (3,3) respectively.

Preamble to Task 3 and Task 4

Tasks 3 and 4 can mostly be completed in any order. It is recommended that you do the easy parts first and leave any hard parts (labelled "advanced") until after you have completed the easy parts.

Task 3: move() method

Add the following method "move" to the class Game:

```
   public void move(int dx, int dy) {
     player.move(dx, dy);
   }
```

Then add a "move()" method to the Player class that moves a player by the given relative distance (dx,dy). That is, if the player is currently at position (x,y), the player should be moved to position (x+dx, y+dy). For example, if the player is currently at position (3,4) and the move method is used with parameters dx=2 and dy=3, then the player should move to position (5,7).

Advanced: (You can try this after completing Task 4)

Add the following lines of code to the move() method in class Game:

```
     player.collect(dot1);
     player.collect(dot2);
     player.collect(dot3);
```

Then add a "collect()" method to the Player class to collect the dot specified as the parameter if that is possible. The player can collect a dot only if the player is at the same position as the dot.

When the player collects a dot, the player's "collectedDots" count should be increased by 1.

When the player collects a dot, the dot should disappear. To implement that, as part of the "collect()" method in the class Player, there should be a call to a "disappear()" method in the class Dot. That method in Dot should contain a single line of code in the body:

```
     exists = false;
```

Task 4: toString()

Add "toString()" methods to each of the classes Game, Player and Dot containing the following code in the body of the method:

- Class Game:
  - `return player + " " + dot1 + " " + dot2 + " " + dot3;`

- Class Player:
  - `return "Player[" + collectedDots + "]" + Util.objectStr(x, y, true);`

- Class Dot:
  - `return "Dot" + Util.objectStr(x, y, exists);`

(Do not vary from the lines of code specified above; a returned string must **exactly** match the above format in order to receive the marks from PLATE.)

Thus a call to the toString() method of class Game returns (but does not print) a string of the form:

```
      Player[0](0,0) Dot(1,1) Dot(2,2) Dot(3,3)
```

The above string indicates the positions of the player and 3 dots, as well as the number of dots collected by the player. Here, [0] indicates the player has collected 0 dots, and each (x,y) indicates the position of a player or dot.

If a player moves to position (2,2) and collects dot2, the string should appear as follows:

```
      Player[1](2,2) Dot(1,1) Dot(-,-) Dot(3,3)
```

Showing the dot’s position as (-,-) indicates that the dot has been collected and has disappeared.

Marking scheme for Part A

Your solution will be marked according to the following scheme:

Task 1: Classes and fields	20 marks
Task 2: Constructors	20 marks
Task 3: move() method	20 marks
Task 4: toString() function	20 marks
Design	15 marks
Indentation	5 marks

Design marks are awarded for the quality of your solution for tasks 3 and 4. More marks are awarded for placing code into the most appropriate classes so as to increase *cohesion* and reduce *coupling*. (To understand coupling and cohesion better, google the words … java tutorial coupling cohesion.)

Indentation marks are awarded for correctly shifting code to the right by one tab between { and } and within if statements.

Note: Once a student's mark is 95 out of 100 or higher (i.e. when expressed as a mark out of 10, it is ≥ 9.5) the mark will be rounded **UP** to full marks. Thus a student should **NOT** spend time trying to get a perfect score of 100/100 (i.e. 10/10).

Note: It is possible to receive partial marks for partially completing a task. The exact formula PLATE uses to award marks for each task is displayed on the PLATE submission page for this assignment, which you can view when you click on the link "Your submission".

YOU WILL HAVE TO IMPLEMENT CORRECT FUNCTIONALITY FOR THE CONSTRUCTORS AND METHODS IN TASKS 1 – 4 IF YOUR ATTEMPT AT PART B IS TO WORK.

HOWEVER, YOU DO **NOT** HAVE TO GET FULL MARKS ON DESIGN AND IMPLEMENTATION IN PART A BEFORE ATTEMPTING PART B.

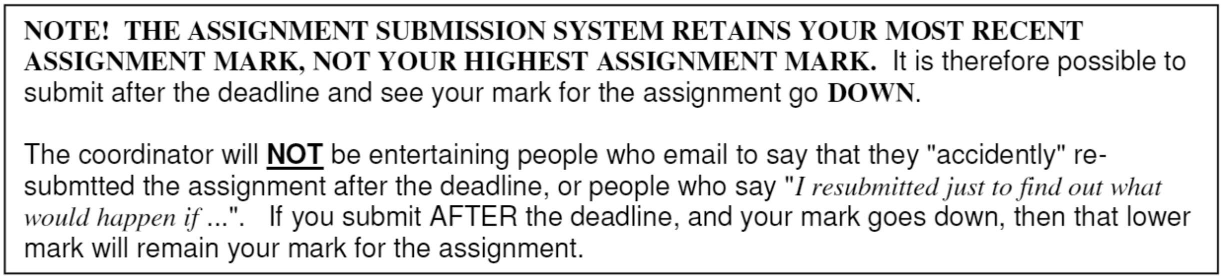
Note: You have to use your solution to Part A to do Part B, so you have to do Part A first. But you do NOT need to score full marks on Part A before doing Part B. You need PLATE to award you a "PASS" for Tasks 1-5, but you do not need the "indentation" marks to start on Part B.

Part B

Due date: 11:59pm on Sunday June 9, 2019 (i.e. 9/6/2019, 11:59pm).
Value: 20%
Topics: Data flow, OO programming basics, Control flow.
Objectives: This assignment supports objectives 1-5.

Introduction

In Part B you will finish the game that you started building in Part A. You will use your solution to Part A as your starting point.



NOTE! THE ASSIGNMENT SUBMISSION SYSTEM RETAINS YOUR MOST RECENT ASSIGNMENT MARK, NOT YOUR HIGHEST ASSIGNMENT MARK. It is therefore possible to submit after the deadline and see your mark for the assignment go **DOWN**.

The coordinator will **NOT** be entertaining people who email to say that they "accidently" re-submtted the assignment after the deadline, or people who say "*I resubmitted just to find out what would happen if ...*". If you submit AFTER the deadline, and your mark goes down, then that lower mark will remain your mark for the assignment.

Note!! Your Part B solution should be submitted on PLATE under the link “Assignment 1 Part B”. Be careful not to submit under the link “Assignment 1 Part A”.

In Part B, you are free to redesign any aspect of your code except for the following:

- Certain class names must be as specified:
  - From Part A: Game, Player, Dot.
  - Introduced in Part B: Enemy, Exit, Main.
- Certain method headers must be as specified:
  - From Part A: Game.move(dx,dy).
  - Introduced in Part B: Game.input() and Game.start(). Also, each object must provide an appropriate toString() function.

Apart from these requirements, you may add, remove or rename fields, add, remove or rename methods, and add classes. Your design choices will be reflected in your design mark (see the section “Marking Scheme” below).

Tasks

Task 1: Exit

Add 1 “exit” object to the game which has the position (4,4). Design the class “Exit” such that its position can be specified by constructor parameters. Your Game’s toString() function should now produce a string in the following format:

```
Player[](0,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4)
```

A player reaches the exit when the player’s position is the same as the exit’s position. Note that the number of dots collected by the player is now represented as [] if no dots have been collected. These square brackets are to be filled with a * symbol for each dot collected. For example, if two dots have been collected, this is represented as [**].

If the player reaches the exit and has already collected all 3 dots, the exit is “opened” and the game’s toString() function displays the exit as “Exit(-,-)” instead of “Exit(4,4)”. That is, the game’s toString() function returns:

```
Player[***](4,4) Dot(-,-) Dot(-,-) Dot(-,-) Exit(-,-)
```

The suggested development sequence for submitting to PLATE is:

1. Define class Exit with appropriate fields and an appropriate constructor, and create the Exit.
2. Define an appropriate toString() function for Exit and submit to PLATE.
3. Modify your program so that the player performs actions in this order: (1) the player moves, (2) the player collects a dot if possible, (3) the player reaches an exit if possible. You may define new methods as appropriate for the task, but the design choice is yours. Submit to PLATE.
4. Modify the Player’s toString() to display collectedDots as a series of * symbols. Submit to PLATE.

Task 2: input() method

Define a method called input() in class Game that takes no parameters. This method should present the user with the following menu:

```
Move (l/r/u/d):
```

This prompt should be printed with a single space following the colon, and should be printed in a way that allows the user to type a movement command on the same line.

If the user types l, r, u or d at the prompt, you should invoke the move() method with appropriate directional arguments that cause the player to move one step left, right, up or down respectively. If the user enters an invalid movement command, you should print the error message “Invalid move”.

HINT: You should probably use a switch statement to implement this movement menu.

The switch statement

The switch statement is more clear than a if/else statement when a variable is compared to a series of different constant values:

If Statement

```
if (place == 1)
{
    System.out.println("Gold medal!");
}
else if (place == 2)
{
    System.out.println("Silver medal!");
}
else if (place == 3)
{
    System.out.println("Bronze medal!");
}
else
{
    System.out.println(
            "Sorry, you didn't place.");
}
```

Equivalent Switch Statement

```
switch (place)
{
    case 1:
        System.out.println("Gold medal!");
        break;
    case 2:
        System.out.println("Silver medal!");
        break;
    case 3:
        System.out.println("Bronze medal!");
        break;
    default:
        System.out.println(
            "Sorry, you didn't place.");
        break;
}
```

An example of using a switch statement to process menu of commands

```
System.out.print("Would you like to quit? (Y/N) ");
String line = keyboard.nextLine();
char answer = line.charAt(0);
switch (answer)
{
    case 'y':
    case 'Y':
        System.out.println("Bye.");
        break;

    case 'n':
    case 'N':
        System.out.println("Excellent!");
        break;

    default:
        System.out.println("Invalid answer.");
        break;
}
```

At this point, submit your solution to PLATE to receive marks for this task. Then continue reading to earn the remainder of the marks.

Task 3: start() method

Define a start() method in class Game taking no parameters. This method should repeatedly print the game's string representation and invoke the game's input() method until the game is over. The game is over when the exit is open. After the game is over, the game's string representation should be printed once more followed on a separate line by the message "You win!".

Define a class called Main with a `public static void main(String [] args)` method. The main() method should ask the user for the initial (x,y) position of the player according to the following sample I/O:

```
Initial x: 2
Initial y: 0
```

The numbers <u>underlined</u> and in **bold** represent input typed by the user. This is "sample" input. Your program should read actual numbers from the user via the Scanner class. The user must not enter negative numbers and is given 3 chances to input a valid number for each coordinate. To handle this, your program must follow the sample I/O below:

```
Initial x: -3
Must not be negative.
Initial x: -1
Must not be negative.
Initial x: 4
Initial y: -7
Must not be negative.
Initial y: -7
Must not be negative.
Initial y: -7
Must not be negative.
Too many errors. Exiting.
```

Note that the user made only 2 mistakes when entering the x position, and the third input was finally accepted. However, the user made 3 mistakes when entering the y position, and so the program was aborted.

NOTE: due to a quirk in the behaviour of Scanner's nextInt() method, you must follow each use of nextInt() by an invocation of nextLine() as shown in the example below:

```
int number = keyboard.nextInt(); keyboard.nextLine();
```

Without adding the extra nextLine(), your program might generate the error "StringIndexOutOfBoundsException"

If the user enters valid initial x and y position, create the Game with the user's chosen initial and (x,y) position for the player, and invoke the game's start() method.

Test your program by running the main() method in BlueJ.

At this point, submit your solution to PLATE to receive marks for this task. Then continue reading to earn the remainder of the marks.

Task 4: Enemy

If the user specifies the player's initial x-position to be 5, the game will be started in "advanced mode". In advanced mode, one more object is introduced into the game called the "Enemy". The Enemy is created at position (5,5) and the game's toString() function displays as follows:

```
Player[](5,0) Dot(1,1) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(5,5)
```

Each time after the player moves one step, the enemy also moves one step in the general direction of the player. If the enemy catches up with the player, the enemy kills the player and the game's toString() functions displays, for example, as follows:

```
Player[*](-,-) Dot(-,-) Dot(2,2) Dot(3,3) Exit(4,4) Enemy(2,1)
```

Displaying the Player's position as (-,-) indicates that the player was killed. In such a case, the main game loop should also terminate and display the message "You lose!" instead of "You win!".

To the user, it will appear as though the player and enemy move simultaneously in each move of the game. However, the computer must still carry out actions in a particular sequence. In this game, the player should be asked to move, collect dots and potentially reach an exit before the enemy is asked to move. If the player reaches an exit and the enemy attempts to kill the player in the same move, the player exits and wins the game without being killed.

An interesting case occurs when the player and enemy are adjacent, the enemy takes one step toward the player and the player takes one step toward the enemy. As a result, the enemy and player swap places and neither before nor after the step do the two coincide at the same position. If not programmed carefully, the player might pass right through the enemy without dying. You need to make sure that the enemy does in this case catch and kill the player in passing.

Each time the enemy is about to move one step, it either decides to continue moving in the same direction that it moved in its previous step, or if that direction will not bring the enemy closer to the player, the enemy will decide to change direction. The enemy will also invoke this decision process on its first step because it has no previous direction to remember.

When deciding to change direction, the enemy considers the distance between the enemy's and player's x positions (the "x distance") and the distance between the enemy's and player's y positions (the "y distance"). If the "x distance" is further than the "y distance", the enemy will change direction toward the player along the x axis (i.e. either left or right). If the "y distance" is further than the "x distance", the enemy will change direction toward the player along the y axis (i.e. either up or down). Otherwise, the enemy will choose a special neutral direction which has the effect of causing the enemy to not move.

This task is complex but it is possible in some cases to receive partial marks for partial completeness. Make sure that you submit regularly to PLATE to see what scenarios are tested and marked first.

At this point, submit your solution to PLATE to receive marks for this task.

Marking scheme for Part B

Your solution will be marked according to the following scheme:

Task 1: Exit	15 marks
Task 2: input() method	10 marks
Task 3: start() method	10 marks
Task 4: Enemy	30 marks
Correct indentation	5 marks
Design	30 marks

Correct indentation means that code should be shifted right by one tab between { and } and within nested control structures such as if statements, switch statements and loop statements. In a switch statement, the code within each case should also be indented. NOTE: You are responsible for your own indentation. Do not rely on BlueJ to indent code for you. BlueJ does not always indent code in the way that is required.

Design refers to how well you have constructed your code to eliminate repeated code, and how well you have structured your program in terms of *coupling and cohesion*. (To understand coupling and cohesion better, google the words … java tutorial coupling cohesion.)

Note: The exact formula PLATE uses to award marks for each task is displayed on the PLATE submission page for this assignment, which you can view when you click on the link “Your submission”.

Note: Once a student’s mark is 98 out of 100 or higher (i.e. when expressed as a mark out of 20, it is ≥ 19.6) the mark will be rounded **UP** to full marks. Thus a student should **NOT** spend time trying to get a perfect score of 100/100 (i.e. 20/20).