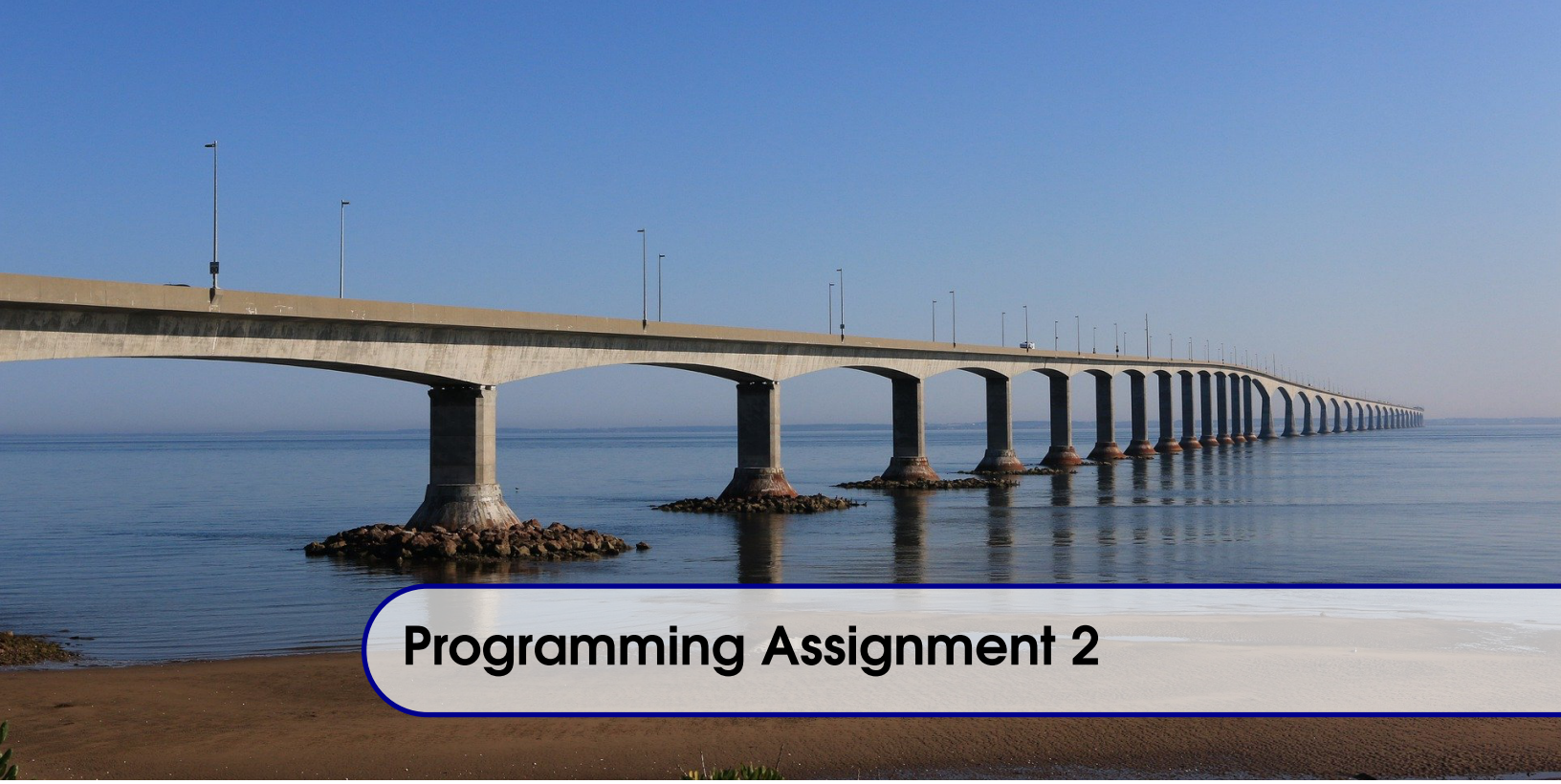

Programming Assignment 2

Assignment Objectives

By completing this assignment you are demonstrating your ability to:

- Apply object oriented programming to encapuslate data.
- Create classes that use inheritance and polymorphism to reuse code.
- Create a program to a testing specification.
- Write code to specifications in Javadoc.

Scenario

For this assignment you will program portions of a simulation for a simplified version of the board game Diamant.

Diamant is a popular bluffing and brinkmanship board game where players press their luck to go deeper and deeper into a mine with the hopes of returning with riches and treasure.

Everything you need to know about Diamant is in this assignment. If you wish to know more information about Diamant please refer to the Board Game Geek Reference page for the game.

Diamant Introduction

Diamant is a board game where players embark on 5 expeditions into a mine to retrieve treasure in the form of rubies. The goal of the game is to collect the most rubies by choosing when to exit the mine at opportune times.

Mine Tiles

The mine is represented by a deck of 35 tiles [1], 20 of which are treasure tiles that have a number between 2-21 representing the number of rubies found on that tile, and 15 trap tiles, where there are 3 of each of the following types: lava, snakes, spiders, boulders, or spikes.

Game Setup

At the beginning of the game all 35 tiles are shuffled and placed face down in a deck so no tiles are revealed. Each player is given a pawn of a particular colour (red, blue, black, green, yellow, purple) and an empty treasure chest.

Players undertake 5 expeditions into the mine within the game.

Expedition

Each expedition starts with an empty treasure pile and a deck of tiles. At the beginning of the expedition all players are considered in the mine until they choose to leave it.

Each expedition consists of a series of rounds that involve the following sequence of phases.

Exploration Phase

A tile is revealed from the top of the deck and the following happens:

1. If the tile has rubies, they are added to the treasure pile.
    - **Example:** The treasure pile has 1 ruby in it. A new tile is drawn for players in the mine. It has 6 rubies. There are now 7 rubies in the treasure pile.
2. If the tile has a trap then the currently revealed mine tiles are checked for traps of the same kind. If there are 2 traps of the same kind in the mine (e.g. 2 lava tiles have been turned over), the mine collapses and all players leave the mine and the treasure pile is lost, with no players currently in the mine get any treasure. Players who are not in the mine when it collapsed (see Decision Phase) retain their treasure and it is safe from loss.
    - **Example:** A spike trap tile is drawn. Checking the mine, that is the first time a spike trap is drawn, and so the game continues to the next phase of round.
    - **Example:** A lava trap tile is drawn. Checking the mine, there was a lava trap tile revealed in a previous turn. The expedition ends, all players remaining in the mine leave the mine, no one gets any treasure from the treasure pile added to their treasure chests, and the treasure pile is set to 0.

Decision Phase

Players decide simultaneously if they want to *stay* in the mine and continue to explore or *leave* the mine. For all players that leave the mine, the number of rubies in the treasure pile are divided between them and the remainder is left in the treasure pile for the players who continue to stay in the mine.

- **Example:** 6 players have entered the mine and there has previously been a tile revealed with 5 rubies, and a tile with 11 rubies, so the treasure pile is at 16 rubies. At the beginning of the next turn, 3 players decide they want to leave the mine. Each player gets 5 rubies for their personal treasure chest, and 1 ruby remains in the treasure pile for the remaining 3 people who chose to stay.

[1] These are sometimes referred to as cards. We have chosen the name tiles to distinguish from traditional playing cards.

Clean Up Phase

After an expedition ends, if 2 trap tiles were revealed causing the mine to collapse, remove 1 of the trap tiles of that type from the revealed from the deck. The now smaller deck is then used in the next expedition.

- **Example:** The mine collapsed due to a second lava trap being revealed. The players then remove on lava trap card from the deck.

If the expedition ends because all players have left the mine, then no card is removed, and the next expedition begins with the deck from the previous expedition (if there was one).

Figure 1: An example of the Diamant board game. The tiles represent the mine, with the purple player showing their treasure chest and a card indicating stay (dark) and a card indicating leave (picture of a camp). Note there is a clear gem, which is a diamond accounting for 5 rubies. For purposes of this assignment we are not implementing diamonds.

Diamant Class Structure

With this assignment, you have been provided with a set of class skeletons with signatures and `javadoc`. The classes in the package `diamant` represent those necessary to represent the specification above.

- `MineTile`: An class containing a set of basic methods for maintaining values from the cards and the current orientation of the cards being either face up or face down.
- `TrapTile`: A tile class that is a subclass of MineTile with specific methods to be overridden.
- `Pawn`: An enumeration for a common set of values for pawn colours available in the game.
- `Player`: An abstract class representing the various players in the game.
- `RiskyPlayer`, `TimidPlayer`, `BalancePlayer`: A set of subclasses of the Player class. Each player has its own strategy for leaving the mine represented in the abstract method leaveMine inherited from Player.
- TileDeck: A representation of a deck of tiles, the key component.

Further there is a class in the base package:

- `DiamantGame`: A class that contains an implementation of Diamant. The test script for this class contained within the main program runs Diamant with 6 players, 2 from each of Risky, Timid and Balance.

Task 1: Implement the Mine (15 marks)

Within the classes above, there are detailed specifications of what should happen for each method of each class. The classes `MineTile`, `TrapTile`, `TileDeck` have had some methods removed and replaced with `/* TODO */`.

Included with all of these classes are a comprehensive test suite for you to test your classes, most of which currently fail. When your class is correct and working to specifications, the test cases will all report with `SUCCESS`.

Task 2: Write Tests (3 marks)

The classes RiskyPlayer, TimidPlayer and BalancePlayer have implementations for the different strategies players can take in the game. Currently, they do not have test suites. Using the examples in the Mine related classes, implement a minimal set of tests that provide complete branch coverage for the class. Branch coverage is defined as running every possible branch in the program from a decision point.

Task 3: Play the Game (2 marks)

You should ensure that the game plays correctly using the DiamantGame implementation. Currently there is a simple test case provided of playing with 6 people.

Constraints

A successful implementation will implement each method in the most general way (i.e. should not return hard coded values), meet the requirements in the method documentation, and pass all of the tests in the test suite.

You need to work backwards using the tests and `javadoc` as a specification, as one would in test driven development, to produce the bodies of the methods. You will need to identify the conditions under which various returns should occur, and what the outputs should look like.

Names of variables should be meaningful and in Java standard camelCase format. All code should be commented thoroughly, and that will count for a small percentage of the mark.

None of the existing code present in the classes methods may be changed. Only the bodies of the methods should be implemented. You may not add any additional instance variables.

If you wish to add private helper methods, please create them directly after the constructor with a very clear indication in the documentation:

```
/**********************************
*  BEGIN HELPER METHODS
**********************************/
CODE HERE
/**********************************
*  END HELPER METHODS
```

```
**********************************/
```

Please also indicate in your code comments where you are calling helper methods in order to help TAs identify them during marking.

Academic Integrity Statement

For this, and all assignments in CS2910, you will include a text file in the project folder, at the same folder level as `src` and `data`, named `academic-integrity-statement.txt`. Within that file you should include the following text:

As a student in CS2910 at the University of Prince Edward Island, I am committed:

- To sustain my effort and engagement for the learning in this course.
- To ask questions about the purpose of or criteria for an assessment if I am uncertain.
- To ask questions about the rules surrounding the assessment if I am uncertain.
- To follow the rules for the assessment, including under conditions of no supervision.
- To treat my personal learning as valuable it its own right.

I submit this assessment, acknowledging the above commitments and that I have followed the rules outlined by my instructor and consistent with the Academic Regulation 20.
At the bottom of this text you should put your name, student number and the date of submission.
<STUDENT NAME>
<STUDENT NUMBER>
<ASSIGNMENT DUE DATE>

Please replace the items in angle brackets (<>) with appropriate information for your assignment.
Please note: We have not included this file in the package download. You must add this file yourself with the contents as specified above. If you do not submit this statement you will receive 0 on the assignment.

Submitting your assignment

Your submission will consist of one zip file containing solutions to all problems (see below). This file will be submitted via a link provided on our Moodle page just below the assignment description. This file must be uploaded by 5pm (Charlottetown time) on the due date in order to be accepted. You can submit your solution any number of times prior to the cutoff time (each upload overwrites the previous one). Therefore, you can (and should) practice uploading your solution and verifying that it has arrived correctly.

An assignment received between 5:01pm on the due date (Friday) and 11:59pm on the following day (Saturday) will be accepted as 1 day late and will have a 10% penalty applied. An assignment received between 12:00am and 11:59pm on the second day following the assignment due date (Sunday) will have a 30% penalty applied. All assignments not received by that date will be considered incomplete and will be given a 0.

What's in your zip file?

A zip file is a compressed file that can contain any number of folders and files. Name your zip file using this format.

Example:
Submission file: `asnX_studentnum.zip`
Where `X` is the assignment number between 1 and 4, and `studentnum` is your student ID number.

Your zip file should contain an IntelliJ project and appropriate files for the assignment and your Academic Integrity Statement.