|  |  |
| --- | --- |
| **ASSESSMENT COVER SHEET** |  |

*Students should add this coversheet, to the start of their assessment before submission through Turnitin.*

|  |  |
| --- | --- |
| **School** | Psychology and Computer Science |
| **Module Code** | CO2402 |
| **Module Title** | Advanced Programming |
| **Assessment Title** | C02402 Advanced Programing Assignment: Snake and Ladders |
| **Module Tutor (if appropriate)** | Miss Aruni Amarasinghe |
| **Course** | BSc Software Engineering |
| **Year of Study** | 2 |

***Academic Misconduct / Plagiarism Declaration***

By attaching this front cover sheet to my assessment, I confirm and declare that I am the sole author of this work, except where otherwise acknowledged by appropriate referencing and citation, or where the assignment brief requires joint working with other students. It also confirms that I have taken all reasonable skill and care to ensure that no other person has been able, or allowed, to copy this work in either paper or electronic form (except where the assessment brief requires co-working with others), and that prior to submission I have read, understood and followed the University regulations as outlined in the Academic Integrity Policy and Procedure for Academic Misconduct available at the following link: <https://www.uclan.ac.uk/study_here/assets/assessment_handbook_2122.pdf>

| **Have you checked the following in order to maximise the grade you can achieve for this assignment?** | **Please select checkbox to confirm** |
| --- | --- |
| Learning Outcomes have been addressed |  |
| Similarity check via Turnitin (and checked the nature of any similarities) |  |
| Referencing accuracy according to provided guide |  |
| Grammar |  |
| Spelling |  |
| Word count (or other length limitation as described in the brief) |  |

**WELLBEING**

|  |  |
| --- | --- |
| We wish to support any student who is experiencing mitigating circumstances which prevents students from performing to the best of their ability when completing or submitting assignments. If you are experiencing such circumstances, then you may apply for Mitigating Circumstances**.** Wherever possible this must be done prior to handing your assignment. | I believe  **that I do**  **I do not need**  to apply for mitigating circumstances for this assignment at this moment in time.  Please **select** as appropriate.  (You may still apply for mitigating circumstances if you subsequently feel that your performance has been adversely affected by issues that you may currently be unaware of). |

**SELF – REFLECTION *This section is suggested for inclusion if appropriate to the assessment otherwise can be deleted***

| **Assessment Criteria**  Details of this can be found in the assignment brief. In order to ensure the assessment process is fair, we want to make sure that the assessment criteria are clear to you in advance. | **Self-Evaluation**  Simply rate how you think this assessment will perform against the assessment criteria; i.e. 1st (very good/excellent), 2:1 (good), 2:2 (competent), 3rd (basic), fail (weak).  This helps us provide detailed comments on your work and clarify things you do not understand |
| --- | --- |
| Relevance | 2:1 |
| Quality of Argumentation | 2:2 |
| Originality | 1st |
| Knowledge and Content | 2:1 |
| Quality of Explanation | 2:2 |
| Style | 2:2 |



STUDENTS DETAILS

|  |  |
| --- | --- |
| UCLan Number: G20844390 | UCL ID: 3000149 |
| Family Name: Weeraman | Given Names: Nethmi |
| Email Address: weeramann@gmail.com | Contact Number: +94 (0)710827460 |

**ASSESSMENTT DETAILS**

|  |  |
| --- | --- |
| Unit Code: CO2402 | Unit Name: Advanced Programming |
| Assessment Title: C02402 Advanced Programing Assignment: Snake and  Ladders | Submission Date: 9/06/2021 |

*Above sections are to be filled by students.*

MARKING DETAILS

|  |  |
| --- | --- |
| Marks Obtained: | Maximum Marks Possible: |
| Comments/Feedback: | |
| Lecturer Name: | Lecturer Signature & Date: |

*Above section is for lecturer’s use only.*

C02402 Advanced Programing Assignment: Snake and

Ladders – Evaluation

Report

***Nethmi Weeraman***

Abstract

In using object-oriented java programming (OOP) to create a command line console game, the methodologies are explored within implementation and described in detail in this documentation. The project, Snake and Ladders is critically analyzed and evaluated as well, using knowledge from the C02402 module.

**Table of Figures**

* Figure 1. Encapsulation showing private methods/variables
* Figure 2. Interface and its implementation for abstraction
* Figure 3. Inheritance shown with square classes
* Figure 4. All overloaded and dynamic getRoll method( ) variance for polymorphism
* Figure 5. Inner class in board
* Figure 6. HashMap to move players
* Figure 7. Javadoc comment

**TABLE OF CONTENTS**

1. Abstract........................................................................................... 4
2. Table of Figures.............................................................................. 5
3. Table of Content............................................................................. 6
4. How the program works
   1. Strengths................................................................................ 7
   2. Weaknesses............................................................................ 7
5. Key java OOP concepts used in game
6. Encapsulation......................................................................... 7
7. Abstraction............................................................................. 8
8. Inheritance.............................................................................. 9
9. Polymorphism........................................................................ 9
10. Other Java concepts used
11. Java Inner class.................................................................... 10
12. Java HashMap...................................................................... 11
13. JavaDoc Comments............................................................. 11
14. References..................................................................................... 12

**How the program works:**

The game is based on the snakes and ladders concept. With players moving up a board to win. Storms, stickies, boosters, snakes, and ladders, for example, are exhibited graphically on the grid, and actions, such as storms, stickies, boosters, snakes, and ladders, are shown. For added interaction, players can choose the number of players in the game and roll dice.

**Strengths:**

* Number of players is customizable.
* Players can have multiple special square activities simultaneously. For instance, using a booster and going up a ladder/down a snake or so forth.
* Players storms and stickies are tracked for each individual. If two players land on a storm square then both get into a storm.

**Weaknesses:**

* End of a snake or ladder isn’t shown on graphical board, only the start is denoted by [S] or [L].
* Error handling could be more prevalent.
* Random number generators are not purely random. The java util random class is based on linear congruential generator, making the die less reliable than a physical one and more prone to bias. ThreadLocalRandom is a suggested better alternative, yet not used in this game (Ruihu, W., 2006).

**Possible Improvements:**

* Better drawn board on console with ladders and snakes drawn in ascii
* Skip round without mentioning what the player rolled

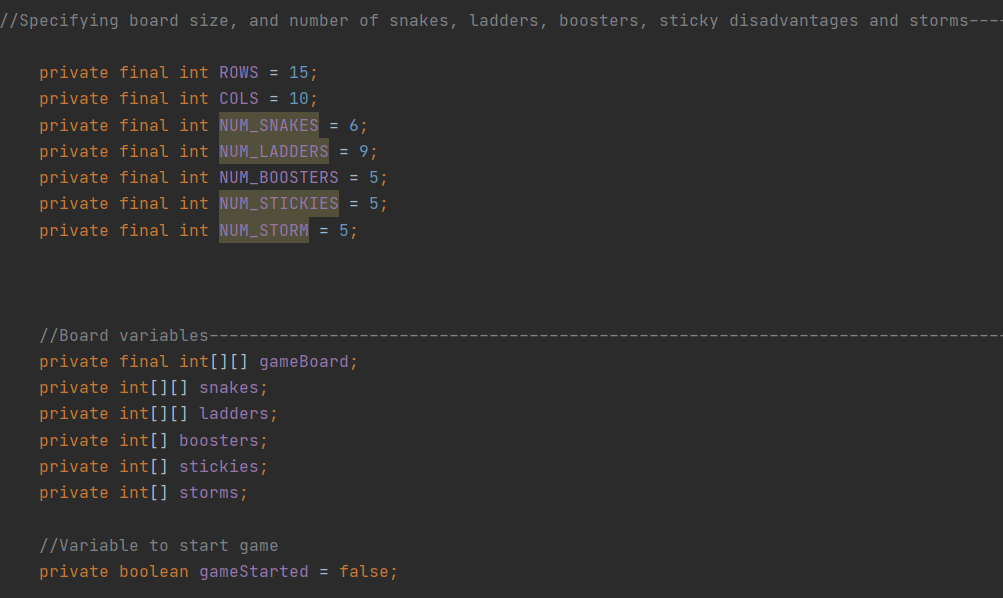
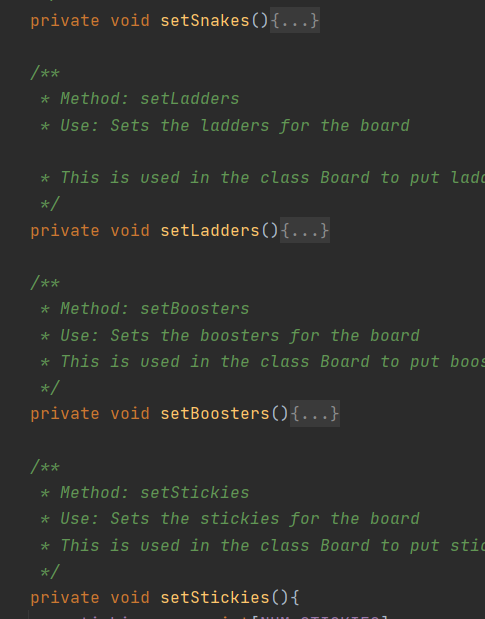
**Key java OOP concepts used in game**

* 1. Encapsulation
  2. Abstraction
  3. Inheritance
  4. Polymorphism

**Encapsulation**

Each program's many elements will attempt to interact with one another automatically. If a programmer wishes to prevent objects from communicating with others, they must be encapsulated in classes. This can be seen in making variable, arrays and methods private, for example in Board class.

This helped in making sure that the methods and variables were not altered from changing or interacting with an object's particular variables and functions, since these methods are not to be changed in the game duration.

Figure 1. Encapsulation showing private methods/variables

**Abstraction**

Abstraction obscures the implementation specifics, attributes and functions could be concealed. For example, to display this, the program uses an interface called RollVariances. This is where the empty method signatures are stored to be implemented by NormalSquares, in turn which is used in StormSquares, StickySquares and so on. This helps in allowing a better look into the different rolls used in a readable manner.

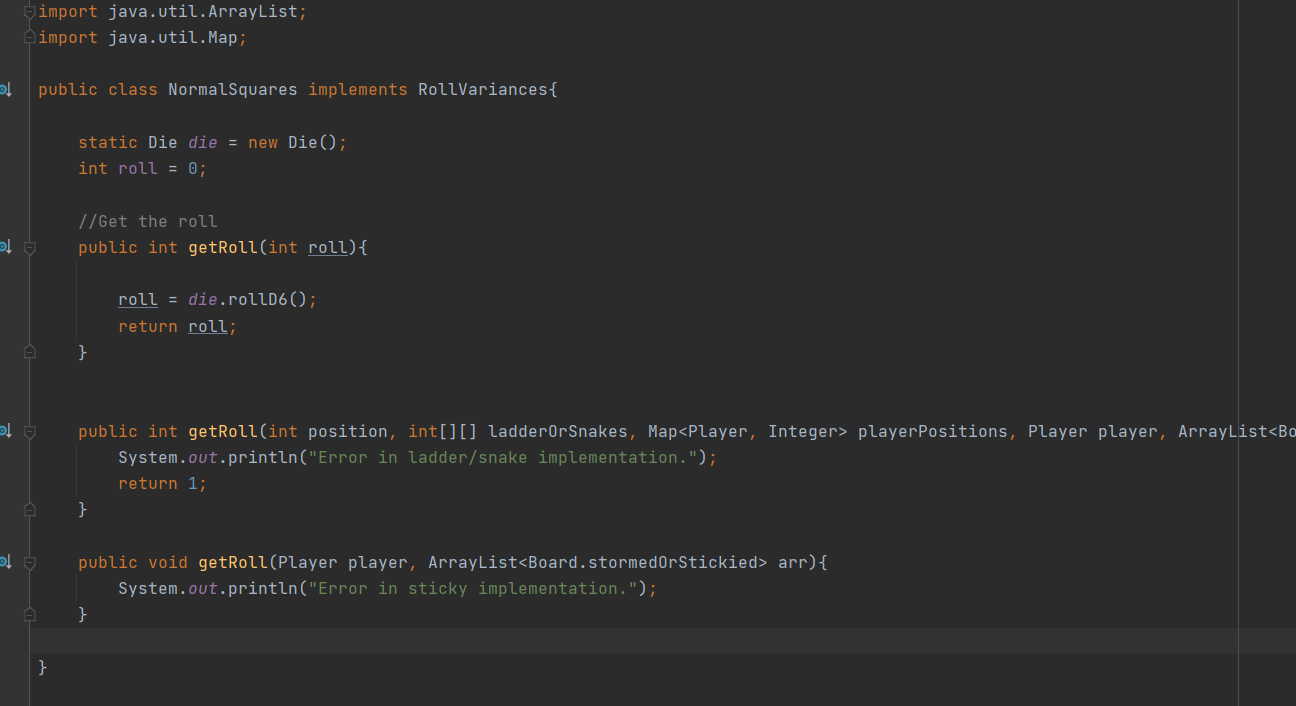


Figure 2. Interface and its implementation for abstraction

**Inheritance**

Inheritance allows one object to use all of the characteristics such as behaviors, attributes and actions of its parent object. This is seen below where the different special squares inherit from the NormalSquares seen above. Below is 2 of the 5 square classes that inherit from normal square class.

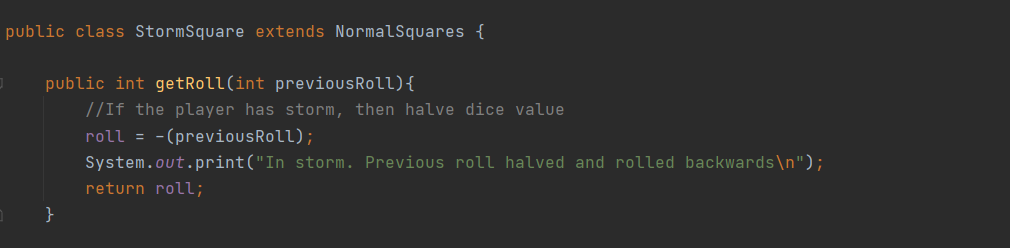
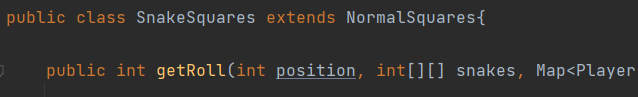
This is done to avoid repetitive code in the system and have uniformity among the special squares and their implementation from a parent square class.

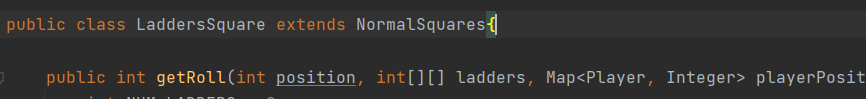
 Figure 3. Inheritance shown with square classes

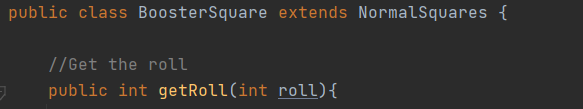
**Polymorphism**

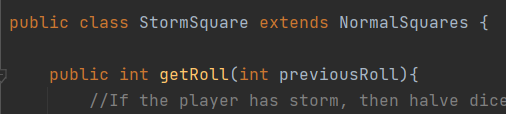
In the static form of polymorphism, called static binding/method overriding, creating numerous methods with the same name but different arguments inside the same class is possible. This is seen with the overloading of the getRoll( ) method. This allows for the less redundancy and avoiding multiple methods having different names for different but similar functionality.

In dynamic polymorphism, a subclass can override a method of its superclass inside an inheritance hierarchy. This allows the subclass to be altered or fully change the method's functionality. This is done with the getRoll( ) method as functionality differs between the different square classes.









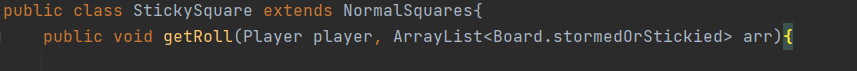


Figure 4. All overloaded and dynamic getRoll method( ) variance for polymorphism

(Kendal, S., 2009.)

**Other Java concepts used:**

**Java Inner class**

It is also possible to nest classes in Java, a class within a class. Nested classes are used to arrange classes that fit together. An inner class can also be static, allowing access without having to create an instance of the outer type. Inner classes makes code easier to understand and manage (Igarashi, A. and Pierce, B.C., 2002. This can be seen with the stormedOrStickied class in board that helps keeps track of the storms and sticky for each individual player.

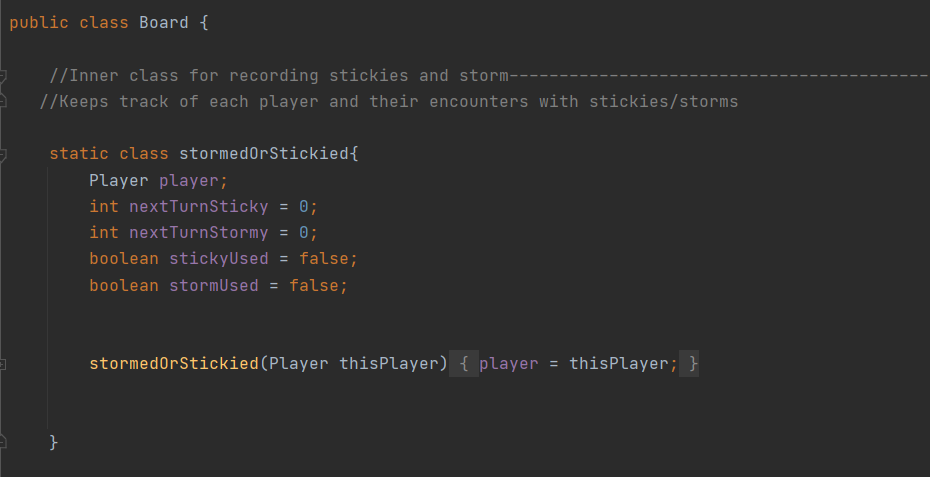


Figure 5. Inner class in board

**Java HashMap**

Items are stored as “key/value” pairs in a HashMap, and retrieving them is done using an index of another data type (e.g. a String). A key (index) is used to link one item to another (value). It has the ability to store many sorts of data (Costa, D. and et al, 2017) Here we store an integer with the key player. This helps to position the players.



Figure 6. HashMap to move players

**JavaDoc Comments**

The syntax of Javadoc comments is fairly similar to that of a standard multi-line remark, with the exception of the added asterisk at the beginning. Any class, function, or field that we want to document can have Javadoc comments added to it.

This Javadoc comment contains a description of the topic on which we're commenting on.and the “@” sign denotes solitary block tags that define special meta-data.usually (Tan, S.H., and et al., 2012).

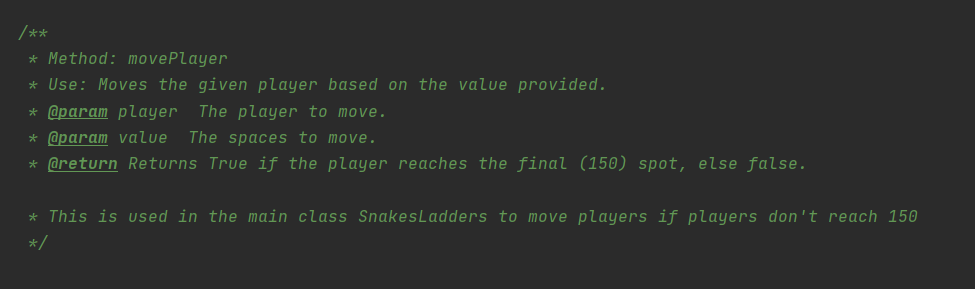


Figure 7. Javadoc comment

(Wu, C.T., 2006)

**References**

1. Kendal, S., 2009. *Object oriented programming using Java*. Bookboon.
2. Wu, C.T., 2006. *An Introduction to object-oriented programming with Java TM*. McGraw-Hill Incorporated.
3. Igarashi, A. and Pierce, B.C., 2002. On inner classes. *Information and Computation*, *177*(1), pp.56-89.
4. Costa, D., Andrzejak, A., Seboek, J. and Lo, D., 2017, April. Empirical study of usage and performance of java collections. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (pp. 389-400).
5. Tan, S.H., Marinov, D., Tan, L. and Leavens, G.T., 2012, April. @ tcomment: Testing javadoc comments to detect comment-code inconsistencies. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (pp. 260-269). IEEE.
6. Ruihu, W., 2006. Generation of Pseudo-random Numbers and its Implementation and Application in Java [J]. *Computer Development & Applications*, *1*.