

SYST 17796 DELIVERABLE 1

BDK GROUP (GROUP 3)

DOUG MEADOWS, KYLE HAINES, BRITTANY LANGLEY

Table of Contents

Team Contract	2
UML Diagram	6
Overview	7
1. Project Background and Description	7
2. Project Scope	8
3. High-Level Requirements	8
4. Implementation Plan	9
5. Design Considerations	9

TEAM CONTRACT

Team Contract

SYST 17796 TEAM PROJECT

Team Name: BDK Group

Please negotiate, sign, scan and include as the first page in your Deliverable 1.

Please note that if cheating is discovered in a group assignment each member will be charged with a cheating offense regardless of their involvement in the offense. Each member will receive the appropriate sanction based on their individual academic integrity history.

Please ensure that you understand the importance of academic honesty. Each member of the group is responsible to ensure the academic integrity of all of the submitted work, not just their own part. Placing your name on a submission indicates that you take responsibility for its content.

For further information, read Academic Integrity Policy here :

<https://caps.sheridancollege.ca/student-guide/academic-policies-and-procedures.aspx>










Team Member Names (Please Print)	Signatures	Student ID
Project Leader: Brittany Langley		991805350
Doug Meadows		991176465
Kyle Haines		991501735

By signing this contract, we acknowledge having read the Sheridan Academic Integrity Policy



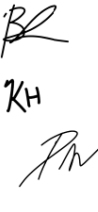
Responsibilities of the Project Leader include:

- Assigning tasks to other team members, including self, in a fair and equitable manner.
- Ensuring work is completed with accuracy, completeness and timeliness.
- Planning for task completion to ensure timelines are met.
- Notifying the professor of any issues in a timely manner so that corrective measures can be taken.
- Any other duties as deemed necessary for project completion.

What we will do if . . .

Scenario	Accepted initials	We agree to do the following (Put an X corresponding to your choice in each box)
Team member does not regularly attend team meetings and/or does not respond to communications in a timely manner.	  	<p>Project leader emails the student citing the concerns and cc's the professor so they are aware of the situation at the very onset <u>X</u> (Mandatory).</p> <p>a) <u> </u> In addition to above, the leader/team will (add your own content here):</p>
Team member does not deliver component on time due to severe illness or extreme personal problem.	  	<p>a) Team absorbs workload temporarily <u>X</u></p> <p>b) Team seeks advice from professor <u> </u></p> <p>c) Team shifts target date if possible <u>X</u></p> <p>d) <u> </u> Other (specify):</p>
Team member has difficulty delivering component on time due to lack of understanding or ability.	  	<p>a) Team reassigns component <u>X</u></p> <p>b) Team helps member <u>X</u></p> <p>c) Team member must ask professor for help <u> </u></p> <p>d) <u> </u> Other (specify):</p>

Scenario	Accepted initials	We agree to do the following (Put an X corresponding to your choice in each box)
Team member does not deliver component on time due to lack of effort.	BL KH PN	a) Team absorbs workload __ b) Team member(s) ask professor to request a Participation Form from <u>all</u> team members. This <i>may</i> result in individualized grades being awarded for a deliverable __ c) Both a. and b. above <u>X</u> d) __ Other (specify):
Team cannot achieve consensus leaving one or more member(s) feeling that their voice(s) is/are not being heard in a decision which affects everyone.	BL KH PN	a) Team agrees to abide by majority vote <u>X</u> b) Team seeks advice from the professor __ c) __ Other (specify):
Team members do not share expectations for the quality of work on a particular deliverable.	BL KH PN	a) Team members will draw on each other's strengths to help bring the quality of the deliverable to a minimal acceptable level <u>X</u> b) Team votes on each submission's quality <u>X</u> c) Team member(s) ask professor to request a Participation Form from all team members, which may result in individualized grades being awarded for a deliverable __ d) __ Other (specify):
Team member behaves in an unprofessional manner, e.g. being rude, uncooperative and/or making one or more		a) Team agrees to avoid use of all vocabulary inappropriate to a business/college setting <u>X</u>

Scenario	Accepted initials	We agree to do the following (Put an X corresponding to your choice in each box)
member(s) feel uncomfortable.		b) Team attempts to resolve the issue by airing the problem at a team meeting <u>X</u> c) Team requests a meeting with the professor to discuss further ____ d) ____ Other (specify):
There is a dominant team member who insists on making all decisions on the team's behalf leaving some team members feeling like subordinates rather than equal members		a) Team will actively solicit consensus on all decisions which affect project direction by asking for each member's decision and vote <u>X</u> b) Team will express subordination feelings and attempt to resolve issue ____ c) Team seeks advice from the professor ____ d) ____ Other (specify):
Team has a member who refuses to participate in decision making but complains to others that s/he wasn't consulted		a) Team forces decision sharing by routinely voting on all issues <u>X</u> b) Team routinely checks with each other about perceived roles ____ c) Team discusses the matter at team meeting ____

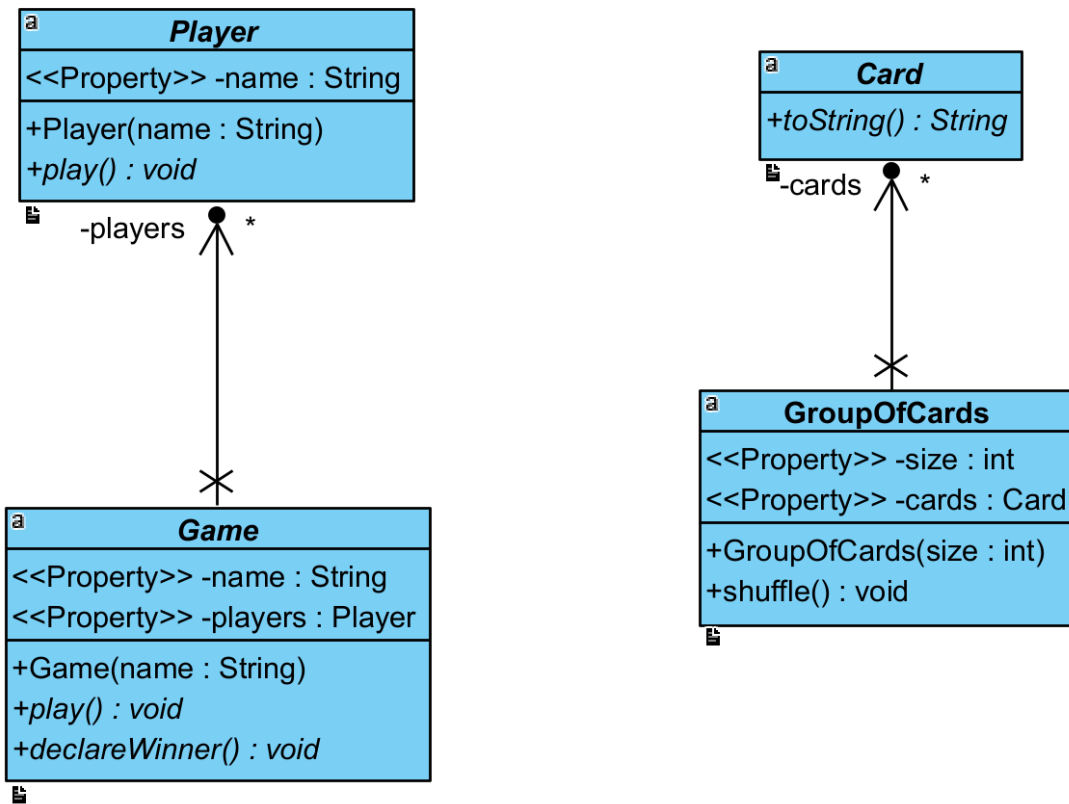
SYST 17796 DELIVERABLE 1

BDK GROUP (GROUP 3)

DOUG MEADOWS, KYLE HAINES, BRITTANY LANGLEY

UML DIAGRAM

Below is the UML diagram for the given base code.



SYST 17796 DELIVERABLE 1

BDK GROUP (GROUP 3)

DOUG MEADOWS, KYLE HAINES, BRITTANY LANGLEY

OVERVIEW

1. Project Background and Description

The goal of this project is to create a Java-based card game application for Blackjack. In the game, a user (player) competes against a dealer (non-user player) to have a hand of cards with a total value as close to 21 as possible without exceeding 21, and to have a higher hand value than the dealer.

How to Play:

1. Initial Deal: The dealer deals two cards to the player and to themselves. The dealer has one card face-up (visible to the player) and one card face-down (not visible to the player). The player can view both cards in their hand.
2. Player Turn: The player can choose to “hit” or “stand”
 - a. Hit: The player is dealt an additional card (this can be repeated so long as the player’s hand value is under 21)
 - b. Stand: Keep the cards in their hand and not receive any additional cards
3. Dealer’s Turn: After the player has chosen to stand, the dealer reveals their face-down card. If the dealer’s total is under 17, they must hit. If the dealer’s total is 17 or higher, they must stand.
4. To Win:
 - a. The player wins if total value of cards in their hand is closer to 21 than the dealer, without busting.
 - b. If either player has exactly 21, they get Blackjack!
 - c. If both player and dealer have the same total hand value, the round is a “push” (tie)
 - d. If the total hand value of either the player or dealer exceeds 21 at any point, they “bust” and lose the round.

Current Base Code:

The current given base code is written in Java following some object-oriented programming principles. The base code is a general framework for card games that can be used to implement specific card games. It contains four classes, three abstract and one concrete. The classes define methods that can be used to handle basic gameplay for a card game.

In the base code, the Player, Card, and Game are abstract classes, and GroupOfCards is a concrete class. Since Player, Game, and Card are all abstract classes, implementations for their methods must be written in subclasses. The Card class also contains a toString() method to be implemented to return a String representation of a card.

The classes are written with private attributes, following the principle of encapsulation, where these attributes can only be accessed or altered through getter and setter methods.

The addition of more classes will be necessary to ensure high cohesion and loose coupling.

2. Project Scope

Technical Scope:

The project is a text-based Blackjack game written in Java, where a user (player) can interact with the dealer to play a game of Blackjack that follows all of the required rules of the game.

The project will be considered complete when:

- All functional requirements and game mechanics are implemented and working as expected
- All features have been tested and all bugs have been fixed
- The game follows the intended flow and structure
- The code has been reviewed by the team and meets standards and design principles
- The final code has been merged with the master branch and pushed to GitHub

Development Team:

Our team consists of three members, Kyle Haines, Brittany Langley, and Doug Meadows. Each team member will be responsible for contributing to game development, working collaboratively on game features and overall system architecture. Each team member will assume an additional dedicated role to divide additional responsibilities.

- Brittany Langley – Project Manager/Team Lead
 - Assume and/or delegate additional tasks, as required
 - Guarantee timelines are met
 - Align team members on objectives
- Doug Meadows – QA Tester
 - Test that the system works as expected
 - Find bugs
 - Verify game rules and logic are correctly implemented
- Kyle Haines – UI/UX Designer
 - Design intuitive user interactions
 - Create engaging interactions for the user
 - Confirm all interactions are user-friendly

3. High-Level Requirements

The Card Game must include the following:

- Player can initiate a new game
- Cards can be shuffled and dealt
- Player can Hit or Stand
- Dealer can Hit or Stand
- Game communicates Win, Loss, or Push as per Blackjack rules.
- Players can know their score
- Player can know Dealer's visible cards
- Players can start a new round or exit after each game

4. Implementation Plan

Git repository URL: <https://github.com/langlbri/Group3Project.git>

Each developer is expected to check in code immediately after any significant updates (such as added functionality, bug fixes, class or method creation, etc.). Upon major updates, each team member will review, pull, and test the code before the changes are merged with the pre-master development branch.

Text files are stored in the **/docs/reports** folder, UML diagrams are stored in the **/docs/uml** folder, and code is stored in the **/src** folder.

Our team will employ the coding standards as outlined below:

1. Each developer will work on their created branch, and push only to their created branch
2. Commits and Pushes will be clearly described with what type of change was made (added method, created class, feature, bugfix, etc.), which file was changed or created, and the date and time of the push.
3. Use of pre-master development branch (named "review") for merges during development and testing, to protect the master branch until code is completed, debugged, and release-ready.
4. Comments in code to be used effectively – avoid overuse of comments, but comment significant changes in code functionality, bugs and bug fixes, or "// TODO" or "// REVIEW" comments to denote where help from the group is needed.
5. Ensure high cohesion and low coupling in all code (each class serves ONE purpose, and classes have minimal dependencies on each other)

Our team will use several tools to complete the Card Game, including NetBeans for writing code, GitHub for collaboration and code management, and Visual Paradigm for creation of UML diagrams.

5. Design Considerations

The current Java base code does follow some principles of object-oriented programming, but there is room for improvement. First, additional classes will be necessary to satisfy high cohesion and loose coupling. For example, Player class should handle all player behaviour logic, the Game class should manage the flow of the game, and a Deck class should be added to implement methods like `shuffle()` and `deal()`.

Specific examples of how the code relates to object-oriented programming principles are outlined below.

Encapsulation

- The current base code uses private attributes in the Player, Game, and GroupOfCards classes to hide the data and protect it through controlled access via getter and setter methods.
- Player class contains **private String name;** which is only accessible through the `getName()` and `setName()` methods in the Player class.
- GroupOfCards class contains a private ArrayList of Card objects, which prevents direct modification by access with `getCards()` method to return the cards in the group of cards, or `setSize()` method which sets the max for the group of cards.
- Game class has **private final String name;** which changes the visibility of the name attribute for the Game to private, and also sets it to final ensuring it is immutable and cannot be reassigned after it is initialized.

Delegation

- The current base code could be modified to allow for helpful delegation
- The GroupOfCards class could be modified to fully represent a deck of cards, so the Game class can delegate methods like shuffle() and deal() to the Deck class.
- The Player class should be implemented such that the Game class delegates player actions to the Player class.

Flexibility/Maintainability

- The current base code is structured using abstract classes containing methods to be implemented by subclasses. This allows for polymorphism, one of the key principles in object-oriented programming.
- For example, with the abstract class Player, a subclass for BlackjackPlayer can easily extend Player class, and define more specific player actions through additional methods.
- The addition of interfaces would allow for improved code flexibility, since classes can only extend one superclass, but can implement multiple interfaces if necessary. For example, a BlackjackPlayer Class could extend Player class as well as implement an interface called BlackjackActions to provide the additional Blackjack player behaviour like hit() and stand() without modifying the Player superclass.
- There is room for improvement in the base code to add more flexibility. As mentioned previously, by implementing a Deck class, and having the Game class delegate deck-related methods and logic to the Deck class, modifications to methods like shuffle() or deal() won't require alterations in the Game class, allowing the Game class manage flow of the game only.