APCS – Svetlik Name: _____

Period #:_____

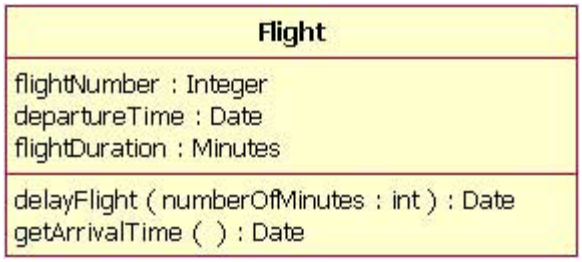
Directions and Design Specifications for Blackjack Application

The Challenge: You are challenged to successfully design and implement in Java a complete Blackjack application according to specifications provided below. The application is phased into eight different parts as follows:

Part I: UML Class Diagrams

A set of UML class diagrams that show you have thought through how you ANTICIPATE the seven parts of the project will be built. UML (Universal Modeling Language) class diagrams are used to illustrate the main essence of a class without going into any detail on the actual implementation of the class. Class diagrams are similar in approach to what is done with Javadocs to show what a class can do instead of how it is implemented, but class diagrams are different in that they are written BEFORE a class is implemented. (Specifically, Javadocs documentation is generated AFTER the code has been implemented).

Two examples of UML class diagrams are shown below:



You are asked to design class diagrams using this format for ALL SEVEN classes that comprise the application.

3 things worth mentioning regarding UML Class Diagrams:

1.) You are not REQUIRED to follow what you create. It is expected that if you do the project right you will end up discovering a lot about the design process that makes your final project significantly different than your original Class Diagram structure. That is normal, and is often how things are done in the industry. Believe it.

2.) You are required to show that you put a "decent amount of thought" into this. This is intentionally vague and subjective. It is usually not difficult to tell when diagrams are done at the last minute, and when they show some serious thought. Make your work representative of your desire to do well on the rest of the project, as this is one in which planning ahead will pay off big-time. Believe it.

3.) YOU EARN 0 POINTS ON EVERY PART OF THE PROJECT YOU TURN IN BEFORE THE CLASS DIAGRAMS. THERE IS NO WAY TO DO THIS PROJECT AND GET POINTS ON ANY OTHER PART WITHOUT FIRST HAVING TURNED IN AN EXCELLENT VERSION OF THIS PART. Believe it.

Part II: Card.java and CardTester.java

A card is the fundamental unit of play, and is used directly and indirectly by every other class in the application. As such, it is arguably the most important piece of code in this project. A card has three fundamental fields (a rank, a suit, and a value). If it turns out to be an ace, its value could be one of two possible numbers (1 or 11) depending on the context.

You should write the Card class alongside a well-built tester that tests every method of the class to show it works well.

→ Continued →

Part III: Deck.java and DeckTester.java

A deck should be implemented to be an array of a pre-determined number of Card objects. In the case of Blackjack, our deck should be built to house 52 Cards.

You should write the Deck class alongside a well-built tester that tests every method of the class to show it works well.

Part IV: Shoe.java and ShoeTester.java

Like a deck, a shoe should be implemented to be an array of Card objects. A shoe consists of one or more decks of Cards. The Shoe class should bring the Cards in from the Deck class.

You should write the Shoe class alongside a well-built tester that tests every method of the class to show it works well.

Part V: Hand.java and HandTester.java

Like a deck and a shoe, a hand also consists of an array of cards. A hand has a value depending on the cards inserted into it.

You should write the Hand class alongside a well-built tester that tests every method of the class to show it works well.

Part VI: Player.java and PlayerTester.java

A player has a Hand and a bankroll, as well as other characteristics to be determined later in class. Instances of this class will interact closely with the Dealer class.

You should write the Player class alongside a well-built tester that tests every method of the class to show it works well.

Part VII: Dealer.java and DealerTester.java

A dealer also has a hand, makes use of a shoe, shuffles the shoe when it is low on cards, and generally does the vast majority of the work necessary to control the flow of the game.

You should write the Dealer class alongside a well-built tester that tests every method of the class to show it works well.

Part VIII: Game.java

This is the main driver of the game. It instantiates a Player and a Dealer, and then calls the Dealer's start() method to get the game going. If done right, this is the simplest class of the whole application.

SEE GUIDELINES BELOW REGARDING FINAL PROJECT SUBMISSION.

Additional Notes

- Thorough Javadocs documentation is required for EVERY class in the application. (Testers do not need Javadocs, though they should be commented when appropriate and necessary to describe what they are testing and how.) This requirement WILL be built into the grade of each individual part of the project.

- Submission for each part of the project should consist of the class, a thorough tester with appropriate output and/or screen shots, and any other parts of the project that were changed along the way. (So, for any class except for the Card class, you need only to turn Card in again if you made any changes to it.) Any changes to previous classes should be clearly-documented in the header of the class as well as in the actual code itself.

- The one exception to this is the FINAL SUBMISSION (specifically for Game.java). This submission will include ALL classes and their testers, as well as ALL documentation generated by Javadocs. It should be neatly-bound (3-hole punched in a folder or binder of some sort.) Treat this like a portfolio-style project.

- The project is open to enhancements… specifically (but not limited to):
  - file saving/retrieval
  - graphics
  - improved (GUI/applet) interface

- HAVE FUN WITH THIS!