COMS 100M Spring 2005
Project 5 – A Game of Twenty-One
Due Thursday, April 14, at 6:00pm

Objectives and background

In this project, you will learn how to use classes and object and how to develop and test code *incrementally*—one class (or even method) at a time. You will implement the functions to play "21," a game similar to Blackjack except there is no gambling of course! The objective of the game is to build a *hand* of 5 or fewer *cards* whose sum is closer to 21 than that of the dealer. If your hand goes over 21 (a *bust*), you automatically lose and the dealer wins. However, you can certainly win the game with a hand less than 21 if the dealer, in trying to beat your hand, goes over 21. In the event of a tie, the game is deemed a draw. Follow the rules of our specialized game of "21" below.

Each card has a value that corresponds to its number and suit. Cards can be numbered as ACE, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, and King. Cards 2-10 have a value equivalent to their face (e.g., a '2' card has the value 2). Jacks, Queens, and Kings all have the value 10 and the Ace card has a value of 1. For example, the hand Ace, 4, 9, Queen has the total value 24 (a losing hand). A round of the game begins with the player (you) being dealt 2 cards and the dealer being dealt 2 cards. The player will see the dealer's hand. From then on, the player can choose to *hit* (add 1 card to the hand) or *stay* (not add cards). The player can hit multiple times before choosing to stay (or until a bust). Deciding to *stay* requires the dealer to then add cards to its hand until its hand has a value over 16 or until all 5 card slots have been used up. The round is over when all of the cards for the player and the dealer have been dealt and the respective hands are tallied to determine a winner. The number of wins and losses must be stored for the player after each round. Below is an example session of the game. *Italics* denote user input. A longer session is included at the end of this document.

```
What is your name? Pat
Hi Pat! Welcome to TwentyOne.

Dealer's hand: QUEEN-H ACE-S  _  _  _
Pat's hand: 3-H 5-H  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 1
Pat's hand: 3-H 5-H QUEEN-S  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 2
---------------------------------------------
Dealer's hand: QUEEN-H ACE-S 7-H  _  _
Pat's hand: 3-H 5-H QUEEN-S  _  _
---------------------------------------------
It's a draw
Would you like to play again? n
Pat won 0 and lost 0 games.
Goodbye
```

The project will involve implementing functions in 5 classes: **Card**, **Hand**, **Player**, **Dealer**, and **TwentyOne**. The skeletons for the code have been provided for you and are available on the course website. You will complete these classes *according to the specifications*. *You must not change any given variable declarations or given method headers.* You may implement additional **private** methods if you wish, but do not implement additional instance variables, public methods, or classes. Also, *do not* use arrays—we will use arrays in the next project to improve our implementation of the game.

Below, we first describe the relationship between the classes and then we explain the purposes and specifications for the individual classes. Read this document carefully. *Work on the classes one at a time*—test each class before moving on to work on the next one. Incremental testing will end up saving you time.

Setup

The starting point of the game is the **main** method in class **TwentyOne** where a **Player** and a **Dealer** are created. The player and dealer will each have a **Hand** of **Card**s. A hand can have at most five cards. The random nature of the game will be implemented through methods in the **Dealer** class.

Now look at the skeleton code we have provided. Notice that the **main** method in class **TwentyOne** is complete but there are three other methods for you to implement. In the other four classes, we have declared all the instance variables and constants and you will need to fill in the method bodies. Notice that the method bodies are empty except for **return** statements that match the return type of the methods. We put these return statements in so that the entire program will *compile*, but of course you cannot play the game yet because the real code of the method bodies are missing. You will need to change these **return** statements as you implement the methods.

First, let's make sure that you can compile the program. Download all the files, including Keyboard, and compile them. Now you can run the **main** method of class **TwentyOne**. You can enter your name and you will see the menu options (hit, stay, or quit). You can even enter your options, but you aren't playing the game (because you have yet to complete the program). Enter option '3' to quit.

This step ensures that you have code that correctly compile and execute. *From this point on, work on the individual classes one at a time. Compile your code often so that you can identify errors soon after they're introduced.* As you work on the individual classes, you probably won't execute the entire program, but still, keep compiling… Since we have given you code that compile to start with, *we expect that your submitted program files will at a minimum compile* (even if the game doesn't run entirely correctly).

The Card class (Card.java)

The **Card** class represents a card in the system. A card has a **number** and a **suit**—these are the instance variables, as declared in the skeleton code. Cards come in four suits: Spade, Club, Diamond, and Heart. The skeleton for the **Card** class includes definitions for the constants that represent each of the suits. Constants are also provided for the *face* cards (Jack, Queen, King, Ace). For example, these constants can be referred to as **Card.HEART** or **Card.JACK**. Use these provided constants, not just arbitrary numbers! Since these constants have **public** visibility, they can (and should) be used in other classes as well. Don't worry about random values here—the **Dealer** class will take care of that later. The **Card** class includes the following methods that you will implement:

- **Card(int number, int suit)**
  - The constructor of the **Card** class. Use this method to create a new **Card** with the specified **number** and **suit**.

- **int getCardValue()**
  - Returns the value for the card according to the following rules:
    - Cards 2 through 10 have a value equal to their face (number)
    - Jacks, Queens and Kings have a value of 10
    - Ace cards have a value of 1

- **int getNumber()**
  - Returns the **number** (1-13) for the card

- **int getGetSuit()**
  - Returns the **suit** for the card.

- **String toString()**
  - Returns a **String** indicating the number and suit of the card. E.g., for a card that is a 3 of Clubs, this method could return: "3C". See example program output.

Testing
You should be compiling your class after writing each method to identify any syntax errors. When you have a few methods completed (or even just one), start testing the class by instantiating several **Card** objects and printing their values. You don't have to wait until you have completed the entire class before testing. How do you test? Write code in DrJava's Interactions Pane or create a main method in another class for testing. Here's an example of some test code written in a separate class. Compile the files and run this **Test** class' **main** method to see if you get the result you want.

```
public class Test{
  public static void main(String[] args) {
    Card c2= new Card(2,Card.CLUB);
    System.out.println(c2.getNumber()); //expect to see 2
    System.out.println(c2.getSuit());  //expect to see 3
    Card h12= new Card(Card.QUEEN,Card.HEART);
    System.out.println(h12.getNumber()); //expect to see 12
    System.out.println(h12.getCardValue()); //expect to see 10
    System.out.println(h12); //expect to see string describing above card
  }
}
```

Make sure your **Card** class is correct before moving on.

The Hand Class (Hand.java)

The Hand class represents a set of **Card**s and will be used by both the **Player** and **Dealer**. The **Hand** class includes 5 **Card** variables to refer to the individual **Card**s in a **Hand**. The other instance variable **handFull** is used to indicate whether the hand is full (has five cards already). At the beginning of a round, the **Hand** does not have (refer to) any **Card**s yet. *Do not* use arrays. The **Hand** class has the following methods that you will implement:

- **Hand()**
  - The constructor for the **Hand** class. This should "clear" all of the card variables to an initial value (HINT: use **null** as the initial value for the **Card** fields) and the hand is not full.

- **boolean isInHand(Card c)**
  - Returns **true** if **Card c** is the same as any **Card** already held in this **Hand**. Return **false** otherwise.

- **boolean addCard(Card c)**
  - Adds **Card c** to the hand in the next available **Card** field. If the Card is not the same as any of the cards already held in the hand, the card should be added and the value **true** returned. If the card already exists in the hand, the function should not store the card and should return **false**.

- **int getValue()**
  - Returns the value of the hand, which is the sum of the values of each of the cards held.

- **void clear()**
  - Clears the cards from the hand by setting all the field (instance variable) values.

- **boolean isFull()**
  - Returns **true** if hand is full, and **false** if hand is not full. This is really a "getter" method for field **handFull**.

- **String toString()**
  - Returns a **String** that represents each of the cards in the hand. This method should make use of the **toString()** method of the **Card** class. See example program output.

This class is tedious to write without arrays, isn't it? This is good motivation to learn about Java arrays for the *next* project.

Testing
Add more code to method **main** of the **Test** class to test your **Hand** class. Below are a few suggestions. You should think of more tests to make sure that your **Hand** class is correct.

```
Hand h= new Hand();
System.out.println(h.isFull());  //expect to see false
System.out.println(h.addCard(c2)); //expect true, Card c2 created previously
System.out.println(h.addCard(h12)); //expect true, h12 created previously
System.out.println(h.addCard(c2));  //expect false, c2 in the hand already
System.out.println(h.getValue());  //expect to see 12
System.out.println(h);  //expect to see string describing the hand
```

The Player Class (Player.java)

The **Player** class represents the player in the system (not the dealer). This class encapsulates the player's hand (set of cards) and is used to store the win/loss history for all of the rounds the player has played. The fields (instance variables) are the player's **name**, its **hand**, and the number of games lost (**gameLost**) and won (**gamesWon**). The **Player** class includes the following methods that you will implement:

- **Player(String name)**
  - The constructor for the **Player** class. This should set the Player's name as specified, create a new **Hand** for the player to use for its rounds of play, and set the number of wins and losses to 0.

- **Hand getHand()**
  - This method should return a reference to the player's **Hand** of **Card**s. This is a simple getter method for the field **hand**.

- **void winHand()**
  - This method will increment the number of games won by the player.

- **void loseHand()**
  - This method will increment the number of games lost by the player.

- **String toString()**
  - This method should return a status message for the player that includes the number of wins and losses since the player started playing. See example output.

Testing
Add more code to method **main** of the **Test** class to test your **Player** class. Below are a few suggestions. You should think of more tests to make sure that your **Player** class is correct.

```
Player p= new Player("Pat");
System.out.println(p.getHand().getValue()); //expect 0, player's hand is
    //empty.  Notice how you can use the Hand method getValue() by first
    //"getting" the player's Hand using the Player method getHand().
p.winHand();  //increment number of wins
System.out.println(p); //expect to see status of player's wins (1) losses(0)
```

The Dealer Class (Dealer.java)

The **Dealer** class represents the dealer in the game. This class encapsulates the dealer's hand (like the player) but also includes functionality to deal cards *randomly* to its own hand or to a player's hand. The single instance variable is the **Deale**r's hand (**dealer_hand**). The **Dealer** class contains the following methods that you will implement:

- **Dealer()**
  - The constructor for the **Dealer** class. This should create a new **Hand** for the dealer to use during the rounds of play.

- **static Card getRandomCard()**
  - This **static** method returns a new **Card** that has a random card number (1-13) and a random suit (0-3, representing Heart, Spade, Diamond, or Club). *Hint*: you need to generate two random integer values, one in [1..13] and the other in [0..3]. At the end of the method, be sure to return a new **Card** with the two generated values.

  Aside: Why is this method a **static** method? This method does not need to access any fields in a **Dealer** object, therefore it doesn't have to be an instance method.

- **static void deal(Hand hand_dealt_to, Hand hand_to_check)**
  - Deal a random card to **Hand hand_dealt_to**. If **hand_dealt_to** or **hand_to_check** already has that card, just get another random card. This method will keep getting random cards until an acceptable one is dealt to **Hand hand_dealt_to**. (*Hint*: Remember the **Hand** methods **isInHand** and **addCard**?)

- **void finishHand(Hand player_hand)**
  - This method should deal cards to the Dealer's hand until the value of its hand is greater than 16 or until all available slots in the hand are used. *Revised April 6th:* Parameter **player_hand** is needed as method **deal** needs to check both the dealer and player's cards.

- **Hand getHand()**
  - Return a reference to the **Hand** of this **Dealer**. Like the Player's getHand, this is a simple getter method for the field **dealer_hand**.

Testing
Add more code to method **main** of the **Test** class to test your **Dealer** class. Below are a few suggestions. Since **Math.random()** is used, you will need to run more/different tests beyond what is shown below to ensure that your **Dealer** class is correct.

```
Dealer d= new Dealer();
//Display 5 randomly generated Cards
for (int k=0; k<5; k++)
  System.out.println(Dealer.getRandomCard());  //using Card's toString()
```

```
//Try to deal some Cards to Player p's hand (p was created in the previous
//set of tests) while also checking what's in Dealer d's hand (nothing)
for (int k=0; k<5; k++) {
  Dealer.deal(p.getHand(), d.getHand());
  System.out.println(p.getHand());
}
```

The TwentyOne Class (TwentyOne.java)

This class is the main class in the system and includes all of the game logic. The skeleton provides the framework and menus for you to use during the execution of the game. First read the provided **main** method and ask for help if you do not understand it. Do not change any code in the **main** method. Then compile all the files and run the **main** method of class **TwentyOne** *before* you implement methods in this class. The careful testing that you have done in the other classes should pay off here and give you error-free compilation. If not, correct the problems in the other classes first before working on class **TwentyOne**. Now implement the following methods one at a time and test each one immediately after its implementation. As you run the **main** method after implementing one method, you expect to see the result of that particular method's functionality.

- **void dealNewRound(Dealer d, Player p)**
  - This method should first clear the hands of both **Dealer d** and **Player p** and then deal two new **Card**s to each person's **Hand**. Display the current **Hand**s. (See example output.)
  - Test-run your program now. You should be able to enter your name and then see both your hand and that of the **Dealer** displayed. Quit the game.

- **boolean doHit(Dealer d, Player p)**
  - (Used when a player wants to *hit*) Deal a **Card** to **Player p**'s hand if it isn't already full, display the new hand, and then check to see if **p**'s hand has busted. If **p**'s hand is already full, print a message saying that a hit is not a valid option and do not deal a card. If **p** goes bust after getting the new card, print a message saying so and return **true**. Otherwise, the game is not over yet, return **false**. The player's game statistics should be updated if necessary. (See example output.)
  - Test-run your program now. Choose *hit* when presented with options. You should see your new hand with a third card along with one of the two following items: the options menu for continuing this round *or* a message saying that you busted. Try another hit or another round. Then quit the game.

- **void doStay(Dealer d, Player p)**
  - (Used when a player chooses to stop adding cards and forces the dealer to finish his hand.) This method finishes dealing to **Dealer d**'s hand, prints both **d** and Player **p**'s final hands, evaluate the hands for a winner, prints a message about the win/loss, and update the player's game statistics as appropriate. (See example output.)
  - Run your completed program now!

Congratulations, you have completed our game "21"! Have fun playing the game!

Submission Instructions

Your submission should include 5 source files: **Card.java**, , **Hand.java**, **Player.java**, **Dealer.java**, and **TwentyOne.java**.

Please ensure that your program compiles all files correctly before submitting. Programs that do not compile will receive a significant penalty.

Example output

```
What is your name? Pat
Hi Pat! Welcome to TwentyOne.

Dealer's hand: QUEEN-H ACE-S  _  _  _
Pat's hand: 3-H 5-H  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 1
Pat's hand: 3-H 5-H QUEEN-S  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 2
---------------------------------------------
Dealer's hand: QUEEN-H ACE-S 7-H  _  _
Pat's hand: 3-H 5-H QUEEN-S  _  _
---------------------------------------------
It's a draw
Would you like to play again? y

Dealer's hand: ACE-C JACK-S  _  _  _
Pat's hand: 8-S 10-S  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 2
---------------------------------------------
Dealer's hand: ACE-C JACK-S 5-S ACE-S  _
Pat's hand: 8-S 10-S  _  _  _
---------------------------------------------
Pat won
Would you like to play again? y

Dealer's hand: JACK-D 7-S  _  _  _
Pat's hand: 6-H 9-S  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 1
Pat's hand: 6-H 9-S 8-H  _  _
Pat busted.
Would you like to play again? y

Dealer's hand: 3-D QUEEN-H  _  _  _
Pat's hand: JACK-C 8-S  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 2
---------------------------------------------
Dealer's hand: 3-D QUEEN-H 6-C  _  _
Pat's hand: JACK-C 8-S  _  _  _
---------------------------------------------
Dealer won.
Would you like to play again? y

Dealer's hand: 5-C 9-H  _  _  _
Pat's hand: ACE-S 9-S  _  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 1
Pat's hand: ACE-S 9-S 3-D  _  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 1
Pat's hand: ACE-S 9-S 3-D 6-S  _

What would you like to do? (1:HIT  2.STAY  3.QUIT) 2
---------------------------------------------
Dealer's hand: 5-C 9-H 2-S QUEEN-D  _
Pat's hand: ACE-S 9-S 3-D 6-S  _
---------------------------------------------
Dealer Busted.
Would you like to play again? n
Pat won 2 and lost 2 games.
Goodbye
```