Objectives and background
In this project, you will learn about using arrays and inheritance in Java. Once again, practice incremental development and testing. The rules of the game “21” are the same as in Project 5, except that you will be implementing a multiplayer game, see example output at the end of this document. The game will be played by three players and a dealer. At the start of the game, the dealer will deal two cards to each of the three players. Then, starting from the first player, each player gets the opportunity to hit or stay. The player can hit multiple times and only after the player “goes bust” or decides to stay will the dealer move to the next player. A player loses the game immediately if its hand busts. After all the players are finished, the dealer deals cards to its own hand until all slots are full or until its hand value is greater than or equal to 17. If the dealer busts, all players who haven’t gone bust win. Otherwise, the dealer’s hand is compared with the players’ hands and the players with more points than dealer (but haven’t busted) win. A player can choose to quit playing at any point.

This game will involve eight classes (but 3+ are given): Card (given), Hand, Deck, Person (given), Player (mostly given), Dealer, TwentyOne (mostly given), and Keyboard (given). A hand is a set of five cards. You will use an array to implement the Hand class, and you will notice that using arrays will make the implementation much simpler than that in Project 5! The player and dealer are both persons, so you will extend the given Person class in implementing the Player and Dealer classes. The dealer will deal from a deck of cards, so you will use an array to implement a Deck class that includes a shuffle function to handle the random feature of the game. Having a deck of 52 cards ensures that no duplicate cards can be dealt, so we no longer need to check for duplicates as we did in Project 5.

Use the provided Card class—do not use your own Card class from Project 5. Implement the classes according to the specifications here. Start fresh from the provided skeleton classes instead of copying code from Project 5. Do not modify any given private instance variable or given method header. You may implement additional private methods if you wish, but do not implement additional instance variables, public methods, or classes.

Be sure to have a new directory (folder) for your Project 6 files that is separate from your Project 5 files. DrJava does not like it if you open two files of the same name from different directories. So if you want to read the code from Project 5 solutions when you’re working on Project 6 files in DrJava, open the Project 5 files in another text editor, e.g., Notepad or Wordpad. Also, remember to turn off DrJava’s backup feature—see 4/6 announcement on the course website.

Setup
The starting point of the game is the main method in class TwentyOne where an array of Players and a Dealer are created. The players and dealer will each have a Hand of Cards. The Dealer fills and shuffles the Deck and will deal from this Deck.

Now look at the completed classes and the skeleton code we have provided. You can compile all the files in this project and run the main method of class TwentyOne. This not-yet-functional game will then prompt you for the players’ names, attempt to display the dealer and player’s hands (no hands will be displayed because the methods are empty!), report that the round of play ends in a draw, and ask you whether to play again. Quit this game. Now you are ready to start writing the code! As for Project 5, work on the individual classes one at a time. Compile your code often so that you can identify errors soon after they’re introduced. 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 is provided for you in this project. Do not modify this class. You should use the provided class rather than the class you developed in Project 5.

The Hand Class (Hand.java)
A hand may contain up to five cards—note and use of the class constant HAND_SIZE, which is set to 5. The instance variable card references an array of five Cards. The instance variable numCards stores the number of cards in the Hand so far. Remember to maintain the definition of numCards throughout your implementation of this class.

The Hand class has the following methods that you will implement:

- **Hand()**
  o The constructor for the Hand class initialize all the fields. This should create the array, but the array does not refer to any Card yet.

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

- **boolean addCard(Card c)**
  o Adds Card c to the hand in the next available Card slot in the array. You do not need to check if Card c already exists in the hand because class Deck (to be completed later) will ensure that there will be not be duplicate cards. Return true if Card c could be added successfully and return false otherwise.

- **void clear()**
  o Empty the hand. This method is called at the beginning of a round of play.

- **boolean isFull()**
  o Returns true if the hand is full. Otherwise return false.

- **String toString()**
  o Returns a String that represents each card in the hand. This method should make use of the toString() method of the Card class. Empty slots in the hand should be marked with the underscore character.

Test your Hand class before moving on. Look at your old test code from Project 5 (or look at the P5 write-up) for inspiration. Save time overall by spending time now in testing your Hand class independent of other classes (except Card, of course).

The Deck Class (Deck.java)
The Deck class represents a standard deck of 52 cards—no duplicates. The instance variable cards references an array of 52 Cards, as its private instance variable. The other instance variable is cardCount, which is a count of the cards in the deck as a round of play proceeds. Remember to maintain the definition of cardCount throughout your implementation of this class.

The Deck class includes the following methods that you will implement:
• **Deck()**
  o The constructor for the Deck class. This should create the array of cards. Initially, the deck should not contain any cards (All slots should be set empty).

• **void fillDeck()**
  o This method should fill the deck with 52 unique cards, 13 cards of each suit. Fill the **Deck** with **Cards** in “ascending order”—you will shuffle the deck later. **fillDeck()** should empty the **Deck** (set **cardCount** to 0) first before filling it (meaning that cards already existing in the deck are discarded).

• **void shuffle()**
  o This method will shuffle a **Deck** (that is already filled, **cardCount** =52). Use the following algorithm for shuffling:

    For $i = 1 .. \text{cardCount-1}$
      Swap the $i^{th}$ card with a card randomly picked from cards numbered from $i$ to $\text{cardCount}$.

  Beware of array indexing! In a Java array the first index is 0, so be careful when you implement the above algorithm.

• **void swap(int i, int j)**
  o This **private** method swaps the **Card** at index $i$ with the **Card** at index $j$ in the **Deck**. Use this method in shuffling the deck.

• **Card takeCard()**
  o This method removes a **Card** from the end of the **Deck** and returns it. Return **null** if the **Deck** is empty. Don’t forget to decrement **cardCount**.

Test your **Deck** class! We have provided a method **showAllCards()** to help you test your **Deck** class. (You won’t use **showAllCards()** in the game.) In the **main** method of this class—we call it a “test harness”—write the following code fragment:

```java
Deck deck = new Deck();
deed.fillDeck();
dead.showAllCards(); //see all 52 cards, ordered
deed.shuffle();
dee.showAllCards(); //see all 52 cards, shuffled
System.out.println(deck.takeCard()); //the last card (52nd card)
deck.showAllCards(); //see all the shuffled cards except the last one
System.out.println(deck.takeCard()); //the second last card (51st card)
```

The idea of a test harness is that each class can have a **main** method to test the functions implemented in that class. Usually you can write a separate class for testing, but since **showAllCards()** is a **private** method—we don’t want any other part of the game to be able to see all the cards—you need to call it from within this class.

**The Person Class (Person.java)**
This class is given. Both **Player** and **Dealer** have some characteristics in common, like they both have a **hand** and a **getHand()** method. The **Person** class is the parent class of classes **Player** and **Dealer**. The only instance variable in the **Person** class is **Hand hand**. There is a constructor and a getter method **getHand()**.
The Player Class (Player.java)

Player is a subclass of class Person. The instance variables declared in class Player are name, gamesWon, gamesLost, and playing. Instance variable playing indicates whether a Player is currently playing in a round. The implementation of class Player is similar to that in Project 5, so all the code that are the same as Project 5 are given. You will need to implement the following methods:

- **Player(String name)**
  - The constructor for the Player class. This should set the Player’s name as specified and set the number of wins and losses to 0.

- **void startPlaying()**
  - Setter method for field playing—set it to true.

- **void stopPlaying()**
  - Setter method for field playing—set it to false.

- **boolean isPlaying()**
  - Returns the play status of the player. A getter method for field playing.

We provided most of the code in this class, but be sure to test the methods you implemented, in particular the constructor, before moving on.

The Dealer Class (Dealer.java)

Dealer is a subclass of class Person. A Dealer has a Deck deck (instance variable) and deals cards from it. The Dealer fills the deck with cards and shuffles it at the beginning of each round. The Dealer class has the following methods that you will implement:

- **Dealer()**
  - The constructor for the Dealer class. This should create a new Deck for the Dealer.

- **void deal(Hand hand_dealt_to)**
  - Take a card from Dealer’s deck and deal it to hand_dealt_to. There is no need to check for duplicates since the cards from one Deck are unique.

- **void finishHand()**
  - This method deals 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.

- **void prepareDeck()**
  - This method should fill the deck and shuffle it.

Test your Dealer class before moving on. See your test code from Project 5 for inspiration.

The TwentyOne Class (TwentyOne.java)

This is the main class from which the game starts. This class is different from Project 5 since this is a multiplayer game. We have provided the main, doHit() and doStay() methods. Do not change the code in these methods. You will implement the dealNewRound() method only. This class implements the game logic, so read the provided methods carefully and make sure you understand it before beginning to code.
void dealNewRound(Dealer d, Player players[])

- This method should first prepare Dealer's Deck for the round. Then it clears the hands of all 
  Persons, sets the Players' play status, deals two Cards to each Person's hand, and displays 
  the current hands. Some of the Players in the array players may be null—this happens if a 
  Player quits in the middle of the game—so you need to check for that before accessing a 
  Player in the array.

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

Final Words
Throughout Projects 5 and 6 we have made many suggestions for testing your code. Testing is done 
incrementally—as you develop the code for the game—and not all at the end. The idea is that you want to 
make sure class X is correct before you write a class Y that depends on X. Sometimes it is tempting to rush 
through coding without stopping to test, but that will burn more time in the long run because you will have to 
deal with many confusing errors in your buggy code when you finally get around to trying to run your 
program. We hope you have learned how to develop some simple test code. Remember incremental 
development and testing in your future work!

Submission Instructions
Your submission should include 5 source files: Hand.java, Deck.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
> java TwentyOne
Welcome to TwentyOne.
What is player #1's name? Huey
What is player #2's name? Dewey
What is player #3's name? Louie

Dealer's hand: 3-D QUEEN-H _ _ _
Huey's hand: 3-C 5-C _ _ _
Dewey's hand: 8-C ACE-D _ _ _
Louie's hand: JACK-C ACE-S _ _ _
Huey's hand: 3-C 5-C _ _ _
What would Huey like to do? 
(1:HIT 2.STAY 3.QUIT) 1
Huey's hand: 3-C 5-C 7-H _ _ _
Huey's hand: 3-C 5-C 7-H _ _ _
What would Huey like to do? 1
(1:HIT 2.STAY 3.QUIT)
Huey's hand: 3-C 5-C 7-H 9-S _
Huey busted.

Dewey's hand: 8-C ACE-D _ _ _
What would Dewey like to do? 
(1:HIT 2.STAY 3.QUIT) 2

Louie's hand: JACK-C ACE-S _ _ _
What would Louie like to do? 
(1:HIT 2.STAY 3.QUIT) 1
Louie's hand: JACK-C ACE-S 10-C _ _ _
Louie's hand: JACK-C ACE-S 10-C
What would Louie like to do?
(1: HIT  2: STAY  3: QUIT) 2

Dealer's hand: 3-D QUEEN-H 9-C
Dealer Busted.
Huey's hand: 3-C 5-C 7-H 9-S
Huey lost.
Dewey's hand: 8-C ACE-D
Dewey won.
Louie's hand: JACK-C ACE-S 10-C
Louie won.

Start another round? y

Dealer's hand: 9-D 8-C
Huey's hand: QUEEN-D KING-H
Louie's hand: 6-S 8-S
Huey's hand: QUEEN-D KING-H
What would Huey like to do?
(1: HIT  2: STAY  3: QUIT) 2

Dewey's hand: ACE-H KING-S
What would Dewey like to do?
(1: HIT  2: STAY  3: QUIT) 3
Dewey quit.
Dewey won 1 and lost 0 games.

Louie's hand: 6-S 8-S
What would Louie like to do?
(1: HIT  2: STAY  3: QUIT) 1
Louie's hand: 6-S 8-S 7-H
What would Louie like to do?
(1: HIT  2: STAY  3: QUIT) 2

----------------------------------------------
Dealer's hand: 9-D 8-C
Huey's hand: QUEEN-D KING-H
Louie's hand: 6-S 8-S 7-H
Huey's hand: KING-D 7-S
What would Huey like to do?
(1: HIT  2: STAY  3: QUIT) 2

Louie's hand: 10-D 2-H
What would Louie like to do?
(1: HIT  2: STAY  3: QUIT) 2

----------------------------------------------
Dealer's hand: 4-D 8-C ACE-D 5-C
Huey's hand: KING-D 7-S
Huey lost.
Louie's hand: 10-D 2-H
Louie lost.

Start another round? n
Huey won 1 and lost 2 games.
Louie won 2 and lost 1 games.

Goodbye