CS100M Grading Guide: Project 5

The coded items below (e.g., c1e, s2a) indicate what a student's solution should accomplish. Codes that begin with the letter 'c' deals with correctness; codes that begin with 's' deals with style.

**Grader:** If a student's solution does not accomplish task c1a, for example, then write the task code 'c1a' along with any diagnostic remarks you can give. Count the number of correctness and style errors separately. In the table below, the top row lists the possible scores (1 to 5). The next row lists the number of correctness errors corresponding to every score category. The style score is determined similarly. Enter the total score (maximum of 10) in CMS as the project score. If there are bonus questions, enter any bonus points separately in the "Bonus Bucket," separate from the project score.

**Student:** Read the grading guide for every project, even if you get a perfect score! Notice from the table below that we often give one or two "freebies," i.e., mistakes that don't cost you any points. Learn from working on the project, and learn from any mistakes.

<table>
<thead>
<tr>
<th>Score</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of correctness errors</td>
<td>Nothing submitted</td>
<td>&gt;15</td>
<td>11-15</td>
<td>7-10</td>
<td>3-6</td>
<td>0-2</td>
</tr>
<tr>
<td>Number of style errors</td>
<td>Nothing submitted</td>
<td>&gt;11</td>
<td>9-11</td>
<td>6-8</td>
<td>3-5</td>
<td>0-2</td>
</tr>
</tbody>
</table>

0. General:
c0a (* to ***): no change to given code (variable declarations and public method headers)
s0a - meaningful variable names
s0b - Code lines are properly indented
s0c - reasonable line lengths, horizontal scrolling should not be required
s0d - well-commented code

1. Card.java:
c1a **: class compiles
c1b – **constructor** and getters (getNumber(), getSuit()) are correct
c1b – **getCardValue()** returns correct value
c1c – **toString()** returns correct string to describe a Card.
s1a – Use the defined constants for the suits and face card numbers (see toString())
s1b – use getter methods (e.g., use **getSuit()** instead of just accessing the field suit). Note: This style guideline is consistent with our discussion in class, even though it is not a universally agreed upon convention. OK here since there're two style error “freebies” (see table above)!
s1c – No superfluous code
s1d – Nice output: face card display is representative of the name. E.g, Jack is shown as “Jack” or “J,” not 11. Similarly, suit name should be representative. E.g., Club is shown as “Club” or “C,” not 3.

2. Hand.java:
c2a **: class compiles
c2b – **constructor** and getter **isFull()** are correct. Constructor body can be empty, can call **clear()**, or can contain the same code as **clear()**
c2c – **clear()**: initialize **handFull** to false, initialize **card1..card5** to null.
c2d – getValue() is correct—must check for null

c2e – isInHand() is correct—must check number and suit explicitly, not just c==card1

c2f – addCard() is correct: add a card that is not a duplicate in the hand to the first available
slot—does not add card if there’re already 5 cards.

c2g - toString() returns correct string to describe a Hand—must check for null

s2a – field handFull is updated in method addCard()

s2b – isFull() uses field handFull, not a conditional statement to check if the hand is full

s2c – No superfluous code

   s2d – print an underscore for an empty card slot in the hand (another marker is ok, leaving it blank is not)

3. Player.java:

   c3a ** – class compiles
   c3b - constructor and getters (getName(), getHand()) are correct
   c3c – winHand(), loseHand() are correct (simple increment)
   c3d – toString() returns both number of games won and games lost.

   s3a – No superfluous code

4. Dealer.java:

   c4a ** – class compiles
   c4b – constructor and getter getHand() are correct. Constructor only instantiates a new
   Hand
   c4c – getRandomCard() returns a new Card with proper random number and suit
   c4d – deal() method loops until it finds a card that can be dealt to hand_dealt_to by using
   the isInHand() methods of both hand_dealt_to and hand_to_check. Note: if Hand
   method addCard() does not check isFull() before adding a card, then the check must
   be done here in deal() or in finishHand() AND TwentyOne.doHit()
   c4e – finishHand() loops to add cards until hand is either full or the hand value is > 17.
       finishHand() needs to check for duplicates in both dealer and player’s hand—need to
       use method deal(), not method addCard(). Note: if Hand method addCard() does
       not check isFull() before adding a card, then the check must be done in deal() above or
       in finishHand() AND TwentyOne.doHit()

   s4a – No superfluous code

5. TwentyOne.java:

   c5a ** – class compiles
   c5b – dealNewRound() clears hands of both Player and Dealer, deals 2 cards to each, and
   displays both hands
   c5c – doHit() checks if Player’s hand is full
   c5d – doHit() checks for bust after card is dealt
   c5e – doStay() calls finishHand() on dealer
   c5f – doStay() checks for bust (dealer), win, or loss properly

   s5a – No superfluous code
   s5b – Code is reasonably compact