AP Computer Science
Dr. Paul L. Bailey

Project 13 - Cards Solutions
Monday, November 28, 2022

Problem 1. State what you understand about the difference between overloading and overriding. Give examples that may or may not exist in the current project.

Solution. Both of these ideas relate to object oriented programming.

Overloading occurs when a class implements more than one method with the same name but different parameter lists.

Overriding occurs when a child class implements a method that had previously been defined in an ancestor class.

In the `Deck` class, we implemented two methods named `deal`, one which dealt one hand, and another which dealt multiple hands. This is overloading.

In the `Card` class, we implemented a `toString` method which overroad the `toString` method from the `Object` class. □

Problem 2. State what you understand about the difference between inheritance and composition. Give examples that may or may not exist in the current project.

Solution. Inheritance is a concept central to object oriented programming.

Inheritance allows one to create a new class which extends an existing class. The new class automatically has all of the global variables and methods as the original class. This produces an ISA relationship between the classes.

Composition occurs when a new class has a variable of an existing class. This produces a HASA relationship between the classes.

We created the `Stack` class which extended the `ArrayList<Card>` class, so that each instance of `Stack` was itself an array list. We could have instead, as a design time decision, created a variable in the `Stack` class of type `ArrayList<Card>`, and used this to store the cards. □

Problem 3. State what you understand regarding Java enumerations.

Proof. An enumeration is a Java type, similar to a class whose instances come from a sequence of fixed constants.

For example, there are exactly four suits of cards, and we created one enumeration to represent them. □

Problem 4. State what you understand regarding Java interfaces.

Proof. An interface is a prototype for a Java class, containing methods with no bodies; a class which implements an interface must contain code for the methods that are outlined in the interface.

For example, in order to sort the stack of cards, we implemented to `Comparable<Card>` interface in the `Card` class, so that other classes could know when one card is less than another. □

Program 1. Create a method `public Hand deal(int n)` in the `Deck` class which deals and returns one hand containing n cards. The cards should come off the top (front) of the deck, and be removed from the deck once they are dealt. Write the source code below.

Solution.

```
public Hand deal(int n)
{
    if (n > size()) return null;

    Hand hand = new Hand();
    for (int i = 0; i < n; i++)
    {
        hand.add(get(0));
        remove(0);
    }
    return hand;
}
```

□

Program 2. Create a method `public Hand[] deal(int n, int k)` int the `Deck` class which deals and returns an array containing k hands, where each hand contains n cards. The cards should come off the top (front) of the deck, and be dealt into the hands in standard card dealing order (one card to the first hand, the next card to the second hand, and so forth, until each hand contains n cards). The cards should be removed from the deck once they are dealt. Write the source code below.

Solution.

```
public Hand[] deal(int n, int k)
{
    if (n * k > size()) return null;

    Hand[] hands = new Hand[k];
    for (int j = 0; j < k; j++)
    {
        hands[j] = new Hand();
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < k; j++)
        {
            hands[j].add(get(0));
            remove(0);
        }
    }
    return hands;
}
```

□