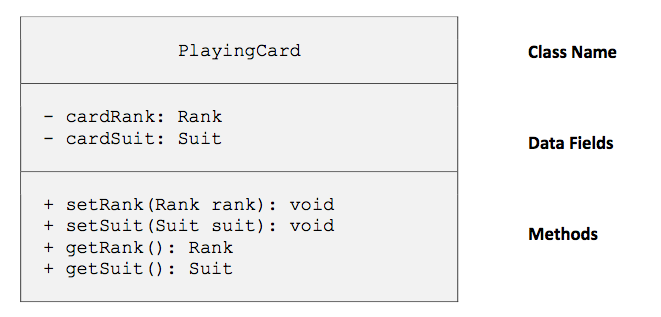
## Model I. (20 min) “Is-a” versus “Has-a”

A large program will often be built out of multiple classes. Often, these classes will have a “has a” relationship. For example, a game program might include three classes: PlayingCard, Suit, and Rank. A PlayingCard object has a Suit and has a Rank, which are handled in a Java program as instance variables.



Less often, you might have two classes which have an “is a” relationship, where one class is a more general class, while the second class is a more specialized version of the original class. A blackjack program might have a BlackJackCard class which is a PlayingCard. Notice the relationship is not symmetrical: every BlackJackCard is a PlayingCard, but not every PlayingCard is a BlackJackCard.

**Critical Thinking Questions**

1. Consider the PlayingCard, Rank and Suit classes. Which two classes are instance variables in the third class?
2. Consider the BlackJackCard and PlayingCard classes.
   1. Which class is the more general class?
   2. Which class is the more specialized class?
   3. Would the four methods from the general class be appropriate in the specialized class?
   4. Give one example of a method that might appear in the specialized class but not the more general class.
3. Explain why the relationship between the PlayingCard, Rank and Suit classes is not an is-a relationship.
4. Consider a Person class (which keeps track of a Person’s name) and a Student class (which keeps track of a Student’s name and student ID number). State the relationship between the two classes as either an is-a or a has-a relationship (similar to the last sentence of each paragraph in the above Model).

1. As a group, come up with another example of a has-a relationship.
   1. Which class is the instance variable?
   2. What is the other class?
   3. State the relationship between your two classes using the phrase “has a”.
2. As a group, come up with an example of an is-a relationship.
   1. Which class is the more general class?
   2. Which class is the more specialized class?
   3. State the relationship between your two classes using the phrase “every \_\_\_\_ is a \_\_\_\_”.

1. Consider the BlackJackCard and PlayingCard classes again, especially your answer to CTQ 2d.
   1. How could this extra method be implementing by adding an instance variable?
   2. Which class would you add the instance variable to?

## Model II. (15 min) Inheritance

The more general class is typically known as the **parent** class, **super** class, or **base** class. The more specific class is typically known as the **child** class, **sub** class, or **derived** class.

The Java keyword extends allows you to indicate that one class should be the child class of the original parent class. The child will then inherit everything from the parent class, including variables and methods.

Create a Person class, Student class and InheritanceDemo program using the starter code from the Student class:

|  |
| --- |
| public class Student extends Person {  private int studentID;  } |

**Critical Thinking Questions**

1. Define the Person class.
   1. Identify the instance variable in the Person class.
   2. Should the variable be declared as public or private?
   3. Create the methods that are in the Person class.
   4. Have they been declared as public or private?
   5. What do the access modifiers (public or private) mean for how the Student class may use the instance variable and methods that it inherits from the Person class?

1. Write and run the main method from InheritanceDemo. Then examine the main method in the InheritanceDemo main method.
   1. Which of the methods identified in part c of the previous question are called in the main method? Give all lines of code here:
   2. What objects are calling those methods? Give the type of each object as well as the variable name.

1. Now consider the Student class.
   1. What instance variable is declared in the Student class?
   2. What methods are missing from the Student class that would usually be present, given that its instance variable is private.
   3. What instance variable has been inherited from the Person class?
   4. What methods have been inherited from the Person class?
   5. Examine the code from the inherited methods in the Person class. Which of these inherited methods are appropriate as-is for the Student class?
   6. Which of the inherited methods are not quite appropriate for the Student class? Explain what is missing from these methods.

**Team Programming:**

In pairs, add a mutator, accessor and toString method to the Student class. Be sure to modify the InheritanceDemo main method, to call your new methods.

## Model III. (15 min) Super

A child class that re-defines a method that was originally defined in the parent class is **overriding** the method. In the team programming exercise that you just completed, the toString method was overridden because both toString methods had exactly the same method header in the Person class and the Student class.

When you override a method in a child class, you can still refer to the overridden method defined in parent class by referencing super instead of this.

|  |
| --- |
| public class Student extends Person {  private int studentID;  public boolean equals(Student object) {  if (super.equals(other)) // call Person equals  return (this.studentID == other.studentID);  else  return false;  }  } |

**Critical Thinking Questions**

1. Examine the above equals method from the Person class. What must be true for two Student objects to be considered equal?
2. How could you rewrite the toString method in the Student class to take advantage of the Person class’ toString method? Give the code here:
3. Carefully compare the equals method header from the Person class to the equals method header from the Student class. Explain why this is actually an example of **overloading**, not **overriding**.
4. How will adding the following method to the Student class change the Student class?

public String getName() {

return super.getName();

}

1. Add a constructor to the Person class. Do not be concerned if the new constructor introduces a compiler error to your other classes.
2. If you were to write a constructor for the Student class, what instance variables should be initialized in this constructor? (Do not attempt to write code for this constructor yet.)

**Group Reflection:**

1. Explain how an is-a relationship differs from a has-a relationship.
2. Review the code that was added to the Student class in this activity. Besides adding the keyword “extends” at the top of the class, what methods were added or modified in the child class? You should include anything that you identified as being inappropriate or insufficient for the Student class, even if it was not edited in the week’s activity.