| **Introduction** | start time: |
|---|---|

| Team Roles | Team Member |
|---|---|
| **Facilitator**: reads the questions aloud, keeps track of time and makes sure everyone contributes appropriately. | |
| **Spokesperson**: talks to the instructor and other teams. Compiles and runs programs when applicable. | |
| **Recorder**: records all answers & questions, and provides team reflection to team & instructor. | |

The facilitator should note the amount of time spent on each Model here:

Model 1:
Model 2:
Model 3:

| ***Model I. (5 min) Inheritance Review*** | start time: |
|---|---|

When developing a large program, we organize our code into classes with methods, instead of writing our code as a few large methods. A utility class full of static methods is a collection of related methods.

But most classes include attributes (instance variables) and non-static methods. Defining a class with variables and non-static methods allows us to create a new, useful object type. Each class definition specifies what attributes the objects will have and what methods can operate on them. Inheritance allows us to share these attributes and methods between similar classes, by organizing them into general classes.

**Critical Thinking Questions**

1. Consider the Person and Student code downloaded from the course repository.
    a. What is the parent class?

    b. What is the child class?

    c. List any attributes that are inherited by the child class from the parent class.

    d. List any attributes that are defined only in the child class.

    e. List any method definitions that appear in BOTH the child class and the parent class.

    f. List any other methods that are inherited by the child class from the parent class. (Constructors are not methods.)

    g. List any methods that are only defined in the child class.

| ***Model II. (17 min) Multiple Types*** | start time: |
|---|---|

The following code illustrates that an object can have multiple types:

- Declared variable type - the type that precedes the variable name in the variable declaration
- Created object type - the type that folllows the keyword `new`

```
public class Polymorphism {
  public static void main(String[] args) {
     Person p;
     Student s;

     Person p1 = new Person("Helen");
     Student s1 = new Student("John");
     Person poly = new Student("Polly");

     System.out.println(p1 instanceof Student);
     System.out.println(s1 instanceof Student);
     System.out.println(poly instanceof Student);
  }
}
```

**Critical Thinking Questions**

6. List the variables that are declared in the above code segment (one per line). For each variable, give its **declared variable type**.

| variable | declared variable type |
|---|---|
| | |
| | |
| | |
| | |
| | |

7. List the objects that have been created in the above code segment by giving the variable name associated with the object (again, one per line). For each object, give its **created object type**.

| **variable** | **created object type** |
|---|---|
| | |
| | |
| | |

8. In the above code, we see our first example where the declared variable type does not match the created object type.
    a. Give that example here:

    b. When the declared variable type does not match the created object type, one type must be a parent (or ancestor) of the other type. Which type is the parent type (the declared variable type or the created object type) in this example?

9. The instanceof operator will return true for checking if the object is checked against either its declared variable type and its created object type. For the above three examples of instanceof code (printed by System.out.println), identify which lines:
    a. Check the object against its declared variable type

    b. Check the object against its created object type

10. The instanceof operator is actually checking for the is-a relationship, i.e., the instanceof operator will return true if the object "is a" type. First *predict* whether any of the following will print something different than the original code (when you replace Student with Person in the "instanceof Student" code). Then replace Student with Person in the three println, and report what each of the following prints:
    a. System.out.println(p1 instanceof Person);

    b. System.out.println(s1 instanceof Person);

    c. System.out.println(poly instanceof Person);

    d. If any of your predictions wrong, explain the correct answer. If all your predictions were correct, explain how you identified which variable (p1, s1, poly) would have different results in the modified code.

11. Given your answer to 8 and 10, explain why the following code is invalid:
    Student s2 = new Person("Chris");

| ***Model III. (17 min) Polymorphism*** | start time: |
|---|---|

***Overriding*** allows a child class to have a different version of a method that it otherwise would have inherited from a parent class.

The different types of a variable help determine which version of the method can and will be used:
- Declared variable type - determines which methods *can* be called (whether the code compiles)
- Created object type - determines which methods are *actually* called (which code is executed)

**Critical Thinking Questions**

12. Identify any methods that are inherited by the Student class from the Parent class.

13. Identify any methods from the Parent class that are overridden in the Student class.

14. Consider the `poly` variable and object from the previous model.
    a. What is its declared variable type?

    b. What is its created object type?

    c. Which methods *can* be called on the poly variable?

    d. Examine the writeOutput method from the Person class. Give the code to call it on the `poly` variable.

e. What method is called by the writeOutput method?

f. Trace which methods are *actually* run (the original method from the parent class or the overridden method from the Student class) by your code in (d).

15. Identify a method that is only in the Student class, not the Parent class.
    a. What is the method?

    b. Give the code to call that method on the poly variable.

    c. What happens when you try to run that code? Explain why.

16. Sometimes we need to call a method from the created object class, even though the declared variable type does not match the created object type. We can do this in three steps:
    a. Write an if-statement that uses the instanceof operator to check if the variable "is a" instance of the child class.

b. Inside the if-statement block, type-cast the variable to a child class object. The first time you do this, you can create a temporary variable from the child class, but it's not necessary.

c. Then call the child class method on the type-cast variable.

d. Give all the Java code here: