

ITSC 1213 - Inheritance Part 3

Introduction

The goal of this lab is to practice working more with inheritance hierarchies and polymorphism.

Concepts covered in this lab:

- Inheritance hierarchies
- Declared type vs actual (run-time) type
- Polymorphism
- Determine object type

Required files or links

- This lab assumes you have successfully completed the Inheritance 1 and 2 labs.

Part A: Polymorphism with ArrayList

Objective: Using an ArrayList with different object types with inheritance and polymorphism

1. Open the NetBeans Project you created in the previous lab. Recall the inheritance hierarchy of `Person` classes from the previous lab and how in part D we were able to create an `ArrayList` of `Person` objects and add objects of type `Student` and `Professor`. This is a core concept in Object Oriented programming which is polymorphism.

One advantage of inheritance is that we can use a superclass reference variable to refer to a subclass object. This allows the reference variables to have different forms, and they are called polymorphic reference variables. This means that if we have an `ArrayList` of `Person` class, we can add `Person` objects, `Professor` objects, or `Student` objects to it. Let's try this.

2. In the main method add a statement to create an `ArrayList` of `Person` type. Name it `contactList`.

3. Add the objects we have created (s1,s2,s3,s4, prof1) into your `contactList` and add a few more objects:

```
ArrayList<Person> list = new ArrayList();

list.add(p);
list.add(s1);
list.add(s2);
list.add(s3);
list.add(s4);
list.add(prof1);

Person p2 = new Person("Elle", "Kambol", 800);
list.add(p2);

Professor prof2 = new Professor("Frank", "Black", 801, "Math", 85000);
list.add(prof2);

Student s5 = new Student("Grace", "Maxeem", 903, "Psychology", 3.4, 95);
list.add(s5);
```

4. Now, add a for-loop below the code you just added. This should loop through the `ArrayList`, calling the `display()` method on each object in the list.
5. Run the project now to see how this works. Note that, even though the objects are all referenced by `p`, which is of `Person` type, the output depends on the type of the object that `p` is pointing to. In Java, this is called polymorphism.

Polymorphism refers to the fact that a superclass reference variable can behave differently depending on what object it is referencing. Java implements polymorphism using two levels of binding. First, at compilation, static binding is performed. Here the compiler matches the method based on the type of the reference variable. In our example, the compiler looks for the `display()` method in the `Person` class. Since `Person` class has this method, static binding passes. Second, at runtime, the Java Virtual Machine (JVM) performs dynamic binding. Here the JVM matches the method based on the type of the object. In our example, the JVM looks for the `display()` method in `Person` class, or `Student` class, or `Professor` class depending on the type of the object.

6. Run your project and fix any bugs you encounter.
7. When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

Part B: Polymorphism with method parameter

Objective: Different object types as method parameter with inheritance and polymorphism

Java allows a superclass reference variable to point to a subclass object. To call a method that takes a superclass type parameter, we may pass a superclass object or a subclass object. Let's try this now.

1. In the main class, create a static method called `showProfile` as follow:

```
public static void showProfile(Person p, int id) { }
```

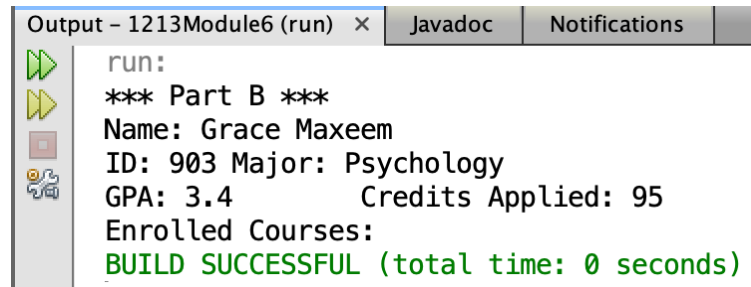
2. Implement this method such that it compares parameter `id` to the object `p`'s `id`. If they match, the method calls the `display` method.

```
public static void showProfile(Person p, int id) {  
    if (p.getId() == id) {  
        p.display();  
    }  
}
```

3. Back in the main method, write a loop to call this method with each element of the `ArrayList` and compare their `id` to 903. If there is a match, call the `display` method to print out the object's details.

```
// module 8 part B  
System.out.println("*** Part B ***");  
  
for (Person person : contactList) {  
    showProfile(person, 903);  
}
```

4. Run your project and fix any bugs you encounter. You will see these lines at the end of your output. This is because Grace's `id` matches the second argument, 903.



```
run:
*** Part B ***
Name: Grace Maxeem
ID: 903 Major: Psychology
GPA: 3.4 Credits Applied: 95
Enrolled Courses:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Note that although the first parameter of this method is of Person class, the method still works if we pass in a Student object or a Professor object.

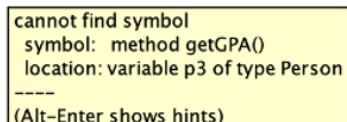
5. When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part C.

Part C: Downcasting and instanceof operator

Objective: Using instanceof to determine if a reference variable is of a specific object type.

1. In Part B, you learned that you can use a superclass reference variable to refer to a subclass object, and that a method call on the variable still works properly if the method is defined in both the superclass and the subclass. What if you want to call a method that is defined in the subclass only? Will that work?

```
Person p3 = new Student("Maya", "Adams", 700, "Music", 3.5, 105);
System.out.println(p3.getGPA());
```



```
cannot find symbol
symbol: method getGPA()
location: variable p3 of type Person
----
(Alt-Enter shows hints)
```

2. If you try to compile your code you will see the following error message:

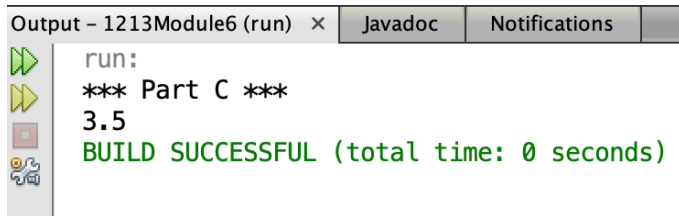


```
/Users/nanaijar/1213Module6/src/pkg1213module6/Module6.java:87: error: cannot find symbol
    System.out.println(p3.getGPA());
                        ^
symbol: method getGPA()
location: variable p3 of type Person
```

3. Why did we encounter this compiler error? This is because, during static binding, the compiler searches for `getGPA()` method in the Person class. However, this method does not exist in Person class. Thus, if you need to call a method that is defined in the subclass only, you have to use a subclass reference variable.
4. To fix this error, you can simply downcast p to a Student type variable by doing so:

```
Person p3 = new Student("Maya", "Adams", 700, "Music", 3.5, 105);
System.out.println(((Student)p3).getGPA());
```

- Run your program again, you should see an output of 3.5

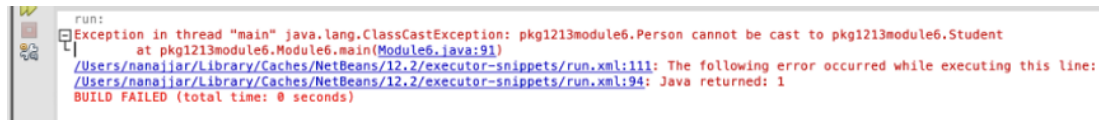


```
Output - 1213Module6 (run) × Javadoc Notifications
run:
*** Part C ***
3.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

- This seems to work well. However, this cast is somewhat dangerous. If you are wrong, and p3 actually points to an object of a type other than Student, then you will encounter a runtime error. Try this by editing your code as follow and run your program again:

```
Person p4 = new Person("Bob", "Lowe", 701);
System.out.println(((Student) p4).getGPA());
```

You will see this type of runtime error:



```
run:
Exception in thread "main" java.lang.ClassCastException: pkg1213module6.Person cannot be cast to pkg1213module6.Student
    at pkg1213module6.Module6.main(Module6.java:91)
/Users/nanajiar/Library/Caches/NetBeans/12.2/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajiar/Library/Caches/NetBeans/12.2/executor-snippets/run.xml:94: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

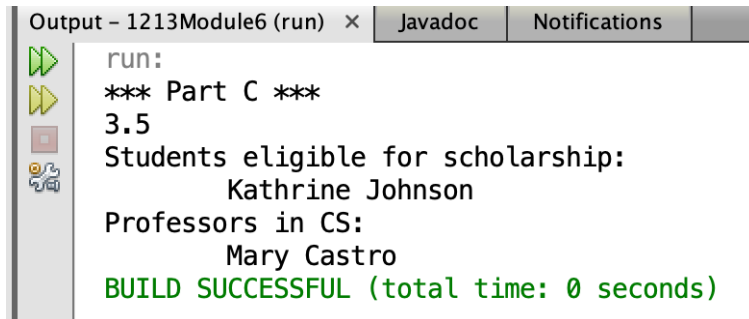
This is because Java cannot convert a Person object to a Student object.

- To protect against bad casts, you can use the `instanceof` operator. It tests whether an object belongs to a particular type. For example, “`p instanceof Student`” returns true if p is referring to a Student object. Modify your code by adding an if statement before casting the reference variable:

```
Person p4 = new Person("Bob", "Lowe", 701);
if (p4 instanceof Student) {
    System.out.println(((Student) p4).getGPA());
}
```

- Now consider the following scenarios:
 - The university wants to offer students a scholarship. The criteria for the scholarship is to have a gpa greater than 3.5.
 - The CS department needs to print out the roster of all professors in the department.
- In your main method, add statements so that it will print out the names (and just the names) of all students who are eligible for the scholarship. Your program should go through the ArrayList that you created in Part A and search for students who satisfy the criteria.

10. Add statements to your main method so that it will print out the name (and just the names) of the professors who work in the CS department. Your program will go through the ArrayList that you created in Part A and search for professors who satisfy the criteria
11. Verify that this works as expected and fix any bugs you encounter. When you are satisfied with your program, show your work to the instructional team to get checked off for this lab.



The screenshot shows an IDE output window titled "Output - 1213Module6 (run)". It contains the following text:

```
run:
*** Part C ***
3.5
Students eligible for scholarship:
    Kathrine Johnson
Professors in CS:
    Mary Castro
BUILD SUCCESSFUL (total time: 0 seconds)
```

So what did you learn in this lab?

1. How to use an ArrayList with different object types with inheritance and polymorphism
2. How to use `instanceof` to determine if a reference variable is of a specific object type.
3. How to use a superclass parameter to allow different object types as arguments using inheritance and polymorphism