CSCI 305 Homework 5

Due Date: April 16, 2018 @ Beginning of Class

Name:_____

Object Orientation

1. Suppose two reference variables `x` and `y` have the declared types `R` and `S` like this:

```
R x;
S y;
```

When the types guarantee that this is safe (i.e., when `S` is a subtype of `R`), Java will allow the assignment `x = y`. When neither of these conditions holds, the assignment might or might not be possible at runtime, and Java will permit it only with an explicit type case, `x = (R) y`. (This kind of type cast is called a *downcast*) With this explicit type cast, the Java language system performs a runtime check to make sure the exact class of y at runtime is in the type `R`.

Suppose the following Java declarations:

```
class C1 implements I1 {
}
class C2 extends C1 implements I2 {
}
class C3 implements I1 {
}
```

where `I1` and `I2` are two unrelated interfaces neither extending the other, and suppose a variable of each type:

```
C1 c1;
C2 c2;
C3 c3;
```

```
I1 i1;
I2 i2;
```

For each of the following assignments say whether Java allows it, disallows it, or allows it only with a downcast, and explain why. (*Hint:* An assignment of `c1` to `i2` is allowed with a downcast, even though the class `C1` clearly does not implement interface `I2`. Think carefully about why.)

- `c1 = i2`
  - **Allowed with downcast, as i2 can be downcast to C2 which IS-A C1**

- `i1 = c1`
  - **Allowed, c1 IS-A I1**

- `c1 = c2`
  - **Allowed, c2 IS-A C1**

- `c3 = i1`
  - **Allowed with downcast, as i1 can be downcast to C3**

- `c2 = c3`
  - **Disallowed, there is no means by which c3 can be cast to a C2**

1. Suppose a derived class `C2` defines a method `m` of type `A2->B2` that overrides a method `m` of type `A1->B1`, inherited from the base class `C1`. Different languages have very different rules about how the types `A1` and `A2`, and `B1` and `B2`, must be related. Investigate and report on this aspect of inheritance, citing the sources you used. Answer the following questions:

   - What is the rule called *covariance*? Give an example of a language that uses the covariant rule. Explain the advantage of this rule.

   Covariance language rules are rules which define a type ordering constraint in the direction of specificity. That is, one type can be replaced by itself or any of its subtypes.

   Covariance is useful in the application of polymorphism, allowing the replacement of a type with its subtypes. It makes sense that a variable of Person could be provided an instance of Student, but that a variable of Student cannot be assigned an instance of Person, as there would be missing methods.

- What is the rule called *contravariance*? Give an example of a language that uses the contravariant rule. Explain the advantage of this rule.

  Contravariance language rules are rules which define a type ordering constraint in the direction of generality. That is, one type can be replaced by itself or any of its supertypes.

  Contravariance is advantageous in cases of Generics. That is it prevents from assigning to a varialbe which expects `List<Person>` something like `List<Student>` as this could lead to methods of the incorrect type.

Parameters

1. The following code fragment uses arrays in Java. The first line declares and allocates an array of two integers. The next two lines initialize it.

```
int[] A = new int[2];
A[0] = 0;
A[1] = 2;
f(A[0], A[A[0]]);
```

Function `f` is defined as:

```
void f(int x, int y) {
    x = 1;
    y = 3;
}
```

For each of the following parameter-passing methods, say what the final values in the array `A` would be, after the call to `f`. (There may be more than one correct answer.)

- By value
  - `A[0] = 0`
  - `A[1] = 2`
- By reference
  - `A[0] = 3`
  - `A[1] = 2`
- By value-result
  - `A[0] = 3`
  - `A[1] = 2` OR
  - `A[0] = 1`
  - `A[1] = 2`
- By macro expansion
  - `A[0] = 1`
  - `A[1] = 3`
- By name

- `A[0] = 1`
- `A[1] = 3`