



COMPSCI 230 Tutorial OO Design Week 4

This is an answer sheet. Answers are marked in RED.

Q1: Remind ourselves of what the following keywords for OO in java mean:

Interface	An interface is like an ordinary class, except with no bodies in the methods.
Abstract	This keyword is used to define an abstract class/methods. An abstract class may include some abstract methods and no abstract methods
Final	The final keyword can be applied to prevent the extension (overriding) of a field, argument, method or class
Super	This keyword can be used to call the parent constructor with no argument. Also if your method overrides one of its superclass' method; you can invoke the overridden method through the use of the super keyword

Q2: What is the result when you run the following code?

```
package q1;

public class Test1 extends SuperClass {

    int x2= 20;
    static int y2 = 20;

    Test1()
    {
        x2 = y2++;
    }
    public int foo2() {
        return x2;
    }
    public static int goo2() {
        return y2;
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SuperClass s1 = new SuperClass();
        Test1 t1 = new Test1();
        System.out.println("The Base object");
        System.out.println("S1.x = " + s1.x);
        System.out.println("S1.y = " + s1.y);
        System.out.println("S1.foo() = " + s1.foo());
        System.out.println("S1.goo() = " + s1.goo());
        System.out.println("\nThe Derived object");
        System.out.println("\nInherited fields");
        System.out.println("T1.x = " + t1.x);
        System.out.println("T1.y = " + t1.y);
        System.out.println("T1.foo() = " + t1.foo());
        System.out.println("T1.goo() = " + t1.goo());
        System.out.println("\nThe instance/class fields");
        System.out.println("T1.x2 = " + t1.x2);
        System.out.println("T1.y2 = " + t1.y2);
        System.out.println("T1.foo2() = " + t1.foo2());
        System.out.println("T1.goo2() = " + t1.goo2());
    }
}
```

Please fill in the output on the next page.

Output:

The Base object

S1.x = 10

S1.y = 12

S1.foo() = 10

S1.goo() = 12

The Derived object

Inherited fields

T1.x = 11

T1.y = 12

T1.foo() = 11

T1.goo() = 12

The instance/class fields

T1.x2 = 20

T1.y2 = 21

T1.foo2() = 20

T1.goo2() = 21

Q3: SuperClass will remain the same, but the Test1 class has slightly changed. What is the output when you run the following code?

```
package q1;

public class Test1 extends SuperClass {
    static int x = 15;
    static int y = 15;
    int x2= 20;
    static int y2 = 20;
    Test1()
    {
        x2 = y2++;
    }
    public int foo2() {
        return x2;
    }
    public static int goo2() {
        return y2;
    }
    public static int goo() {
        return y2;
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SuperClass s2 = new Test1();
        System.out.println("\nThe static Binding");
        System.out.println("S2.x = " + s2.x);
        System.out.println("S2.y = " + s2.y);
        System.out.println("S2.foo() = " + s2.foo());
        System.out.println("S2.goo() = " + s2.goo());
    }
}
```

Output:

```
The static Binding
S2.x = 10
S2.y = 11
S2.foo() = 10
S2.goo() = 11
```

As a bonus: re-add the println/variable commands from the original Test1 to the main() and see how the result changes. These print statements are included in the answer code.

Q4: In which class is the method s2.goo() called? Hint: Run the debugger and step through code.

It calls goo() from the SuperClass.

Q5: What is the static type of variable s2?

Its static type is of type SuperClass

Q6: Are we able to make a call to method foo2() from variable s2?

Not initially but yes we are able to with casts. Because s2 is of dynamic type Test1, we can typecast it to Test1 and then call foo2(). ((Test1)s2).foo2(); We can even create a var with static type Test1 and cast s2 to it : Test1 t_s =(Test1)s1;

Q7: Test1 t2 = new SuperClass();

What is the result from the above line of code?

Compile-time error. Type mismatch: cannot convert from SuperClass to Test1

Q8: Test1 t2 = (Test1) new SuperClass();

What is the result from the above line of code?

Run-time error. Compiler type-checking lets this pass but an exception is thrown at runtime of type ClassCastException.

Q9: what is the default visibility for constructor?

Package-private. It is visible only within its own package

Q10 Look at the skeleton code Test2.java. You will see an error message “The constructor SuperClass() is not visible”, How to solve this problem?

Change the SuperClass’s constructor to protected. E.g.

```
protected SuperClass() {  
    x = y++;  
}
```

Q11: How many of the Interface’s methods must we provide an implementation for?

All methods defined in the Interface must be implemented in the Class

Q12: When attempting to add a new method definition to an Interface A, which is used by Class X, it is preferred that the Interface A is extended by another Interface: B. Why is this preferred?

Because a class that implements Interface A must provide implementations for every method defined by A. Adding a new method to A will break every class that previously implemented A before adding a new method. The reason is that the classes do not provide an implementation for the new method. By extending A inside Interface B, we can add the new method without creating errors in any class that implements A. For example, look at the Jdocs for MouseListener to see how many implementing classes you could break!

Q13: How many Interfaces can be implemented by one class?

As many as you like. There is no limit on how many Interfaces you may implement at one time.

Q14: Abstract classes are similar to interfaces in that we don't have to provide method implementations. However, we are able to provide implemented methods inside an abstract class. How do you define a method in an abstract class for which you don't provide an implementation?

Use the abstract keyword in the method definition. `public abstract ... etc`

Q15: Can abstract methods be defined in a concrete (Non-abstract) class?

No. If a class defines any abstract methods, the class itself must be abstract.

Q16: How do you extend an abstract class A and implements Interfaces B,C, and D in Class SubClass1 at the same time?

```
public class SubClass1 extends A implements B,C,D
{
    //Rest of the code is omitted.
}
```

`extends` comes first and `implements` follows by convention.

Q17: Generics is a feature of the Java programming language that allows type safety. The Java Collections Framework provides some data structures such as List, ArrayList, and LinkedList that can be used with Generics.

If you created a ArrayList (`List<String> list = new ArrayList<String>();`) using String as the generic, What kind of error would you get if you attempted to place an object that was not of type String into the ArrayList: Run-time or Compile-time?

Generics is compile-time type checking. It would result in a compiler error.