Topics: Inheritance, abstract classes, interfaces, and static vs. dynamic type

1 Inheritance (high level)

- Allows specialization or modification of a class
- Allows a class to retain certain properties of another class

2 Inheritance (Java)

- ... <classname> extends <existingclass> { where <classname> is the class being declared as a subclass (a child) of <existingclass> (the superclass or parent)
- All classes can be extended (except those declared as final), even classes from the Java standard library
- When a class inherits from another class, all fields and methods declared in the parent class as public or protected are accessible through the child class
- Methods declared in a parent class may be overridden, changing the behavior of the method

Example

```java
public class Person {
    public int age;
    public String name;
}

public class Worker extends Person {
    protected int money;

    public void work() {
        age++;
    }
}

public class Proletarian extends Worker {
    public void payMoney() {
        money--;
    }

    public void work() {
        age += 2;
    }
}

public class Capitalist extends Worker {
    public void work() {
        age--;}
```
public void makeMoney(Proletarian p) {
    p.payMoney();
    money++;
}

3 Static type vs. Dynamic type

- The static type of a variable is the type used upon declaration
- Static type is known at compile time (before you run the code)
- The dynamic type of a variable is the type it actually is during execution
- Dynamic type can be determined by examining the object a variable references during execution
- Variables with static type T may have dynamic type T or any subtype (subclass) of T
- Assignment of a subtype to a supertype is allowed without a cast, but assignment from a supertype to a subtype is allowed only with a cast (and can possibly cause a ClassCastException)

Example

Worker w = new Worker(); // static and dynamic type Worker
Proletarian p = new Proletarian(); // static and dynamic type Proletarian

w = p; // now has static type Worker, dynamic type Proletarian

Proletarian p2; // static type Proletarian
p2 = (Proletarian) w; // cast needed because w has static type Worker

4 Object and its methods

- The class Object is the top of the inheritance hierarchy – all classes that do not extend another class are automatically subclasses of Object
- Object contains useful methods such as toString() and equals(), which, due to inheritance, are available in all classes (and may be overridden in them, as you have seen)

5 Interfaces

- An interface is a definition similar to a class in which methods do not have implementations and all fields are final and static
- The keyword to extend an interface is implements
- Interfaces cannot be instantiated

Example
public interface Revolutionary {
    public String incite();
}

public class Proletarian extends Worker implements Revolutionary {
    public void payMoney() {
        money--;
    }

    public void work() {
        age += 2;
    }

    public String incite() {
        return "A dictatorship of the proletariat!";
    }
}

public class Capitalist extends Worker implements Revolutionary {
    public void work() {
        age--;
    }

    public void makeMoney(Proletarian p) {
        p.payMoney();
        money++;
    }

    public String incite() {
        return "A dictatorship of the bourgeoisie!";
    }
}

6 Announcements

- Reading: Programming Into Java Ch 4-6

7 Abstract classes

- An abstract class is a class in which some methods may be left undefined
- Just like interfaces, abstract classes may not be instantiated
- Methods left unimplemented are declared with the abstract keyword, just as the class itself is
- Abstract classes can be extended just like regular classes

Example

public abstract class Worker extends Person {
    protected int money;
public abstract void work();
}

8 Peer-instruction question

What happens if I try to compile and execute the following?

[0] Revolutionary r;
[1] Proletarian p;
[2] Worker w;
[3] Capitalist c;

[4] p = new Proletarian();
[5] c = new Capitalist();
[6] r = p;
[7] w = c;
[8] r = (Revolutionary) w;
[9] p = (Proletarian) r;

White: It will compile and execute without error
Blue: There will be a compile-time error at [6]
Green: There will be a run-time error at [6]
Pink: There will be a compile-time error at [8]
Yellow: There will be a run-time error at [9]