Inheritance

Code reuse is a goal of object-oriented programming.

Inheritance implicitly copies code one class to another.

New class only codes the difference.

Add a new method: extends the inherited class.

Redefine an inherited method: alters its behavior.

Mechanics

To inherit `A`'s code into `B`:

```
public class B extends A {
    ...
}
```

Then

```
B b = new B()
```

creates an object with all of `A`'s methods and data, as well as any added by `B`.

Method dispatch

Method dispatch after `B b = new B()`:

1. `b.foo()` looks in class `B` to see if `foo` is defined there. If so, it is executed.
2. If not, it looks in `A` (because `B` extends `A`), and executes it if it is there.
3. Whichever `foo` executes, if it make a method call, the same process occurs
   (search `B` before `A` for called method).

Example

```
public class A {
    public void foo() {
        System.out.println("foo");
    }
}

public class B extends A {

}

public class Main {
    public static void main(String[] args) {
        B b = new B();
        b.foo();            // "foo" gets printed
    }
}
```

Example

```
public class A {
    public void foo() {
        bar();
    }
    public void bar() {
        System.out.println("A:bar");
    }
}

public class B extends A {
    public void bar() {
        System.out.println("B:bar");
    }
}

public class Untitled {
    public static void main(String[] args) {
        B b = new B();
        b.bar();            // "B:bar" gets printed
        b.foo();            // "B:bar" gets printed
    }
}
```

Small print

1. Superclass constructor is called automatically.
2. To call a different superclass constructor: `super(argument list)`.
3. To call superclass's overridden method: `super.methodname()`.
4. Can't access superclass `private` elements (see `protected`).

Example: Simple list of strings

```
public class StringList {

    private String list[];
    private int numElements;

    public StringList() {
        numElements = 0;
        list = new String[100];
    }

    public void add(String s) {
        ... code to add s to the end of the list ...
    }

    public int indexOf(String s) {
        ... code to report the first occurrence of s ...
    }

    public String toString() {
        ... code to create a String representation of the list ...
    }

    ... etc ...
}
```

Example: Simple list of (sorted) strings

```
public class SortedStringList extends StringList {

    public void add(String s) {
        ... code to add s into its sorted position ...
    }

}
```

Example: File Processor

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class ProcessLines {

    public void process() {
        Scanner file = null;
        Scanner in = new Scanner(System.in);
        System.out.print("File: ");
        String name = in.nextLine();
        try {
            file = new Scanner(new File(name));
        }
        catch (FileNotFoundException e) {
            System.err.println("File not found");
            return;
        }
        while (file.hasNextLine()) {
            System.out.println(processLine(file.nextLine()));
        }
    }

    public String processLine(String s) {
        return s;
    }
}
```

Example: Using File Processor

We can now create other programs that process lines using inheritance and just a few lines of code.

```
public class ToUpper extends ProcessLines {
    public String processLine(String s) {
        return s.toUpperCase();
    }
}


public class Main {
    public static void main(String[] args) {
        ToUpper x = new ToUpper();
        x.process();
    }
}
```

Example: Practice

```
public class E extends F {
    public void method2() {
        System.out.print("E 2 ");
        method1();
    }
}
```

```
public class F extends G {
    public String toString() {
        return ("F");
    }
    public void method2() {
        System.out.print("F 2 ");
        super.method2();
    }
}
```

```
public class G {
    public String toString() {
        return ("G");
    }
    public void method1() {
        System.out.print("G 1 ");
    }
    public void method2() {
        System.out.print("G 2 ");
    }
}
```

```
public class H extends E {
    public void method1() {
        System.out.print("H 1 ");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        G o = new G();                          // Try for E, F, G, H
        System.out.println(o);
        o.method1();
        System.out.println();
        o.method2();
    }
}
```

Critters!

In lab we'll use a class called `Critter`. We'll use five of its methods.

```
public boolean eat()                    // Default: false
public Attack fight(String opponent) // Default: Attack.FORFEIT
public Color getColor()                 // Default: Color.BLACK
public String toString()                // Default: "?"
public Direction getMove()              // Default: Direction.CENTER
```

```
Attack.ROAR, Attack.POUNCE, Attack.SCRATCH, Attack.FORFEIT
```

```
Direction.NORTH, Direction.SOUTH, Direction.EAST,
    Direction.WEST, Direction.CENTER
```

Critter simulator

1. A critter simulator initializes a world with various critters and food
2. Each unit of time the simulator calls your methods to determine your critter's behavior.
3. Your critter class should inherit the default behavior and override what needs to change.
4. If your critter needs to remember anything between time units, make data fields and a constructor.

Example from an old exam

Write a class Wanderer that extends Critter. Each Wanderer is represented as a blue "W" and begins walking north. Each time a Wanderer encounters food it, with equal likelihood, either eats it and turns right or doesn't eat it and continues walking in the same direction.

```java
import java.util.Random;
import java.awt.Color;

public class Wanderer extends Critter {

    private Direction curDir;
    private Random rand;

    public Wanderer() {
        curDir = Direction.NORTH;
        rand = new Random();
    }

    public String toString() {
        return "W";
    }

    public Color getColor() {
        return Color.BLUE;
    }

    public Direction getMove() {
        return curDir;
    }

    public boolean eat() {
        if (rand.nextInt(2) == 0) {
            // Don't eat
            return false;
        } else {
            // Eat
            if (curDir == Direction.NORTH) {
                curDir = Direction.EAST;
            } else if (curDir == Direction.EAST) {
                curDir = Direction.SOUTH;
            } else if (curDir == Direction.SOUTH) {
                curDir = Direction.WEST;
            } else {
                curDir = Direction.NORTH;
            }
            return true;
        }
    }

}
```