Polymorphism
Motivation

- **Given the following:**
  
  ```java
  Lawyer laura = new Lawyer();
  Marketer mark = new Marketer();
  ```

- **Write a program that will print out the salaries and the color of the vacation form for each employee.**
Polymorphism

- A reference variable of type T can refer to an object of any subclass of T.
  
  ```java
  Employee person = new Lawyer();
  ```

- **polymorphism**: The ability for the same code to be used with several different types of objects and behave differently depending on the type of object used.
The variable person has two types:

- **static type**: Employee
  - This is the type that the compiler uses to determine if statements are legal Java statements.
  - Therefore, any method called with the person variable must be declared in the Employee class (or else the compiler will complain).

- **dynamic type**: Lawyer
  - This is the type that the Java virtual machine uses to execute code when the program is run.
  - Any method called with the person variable will execute the version of that method defined in the Lawyer class.
Properties of polymorphism

Employee person = new Lawyer();
System.out.println(person.getSalary());        // 40000.0
System.out.println(person.getVacationForm());  // "pink"

- You can call any method from Employee on the person variable, but not any method specific to Lawyer (such as sue).

- Once a method is called on the object, it behaves in its normal way (as a Lawyer, not as a normal Employee).
Polymorphism and parameters

```java
public class EmployeeMain {
    public static void main(String[] args) {
        Lawyer laura = new Lawyer();
        Marketer mark = new Marketer();
        printInfo(laura);
        printInfo(mark);
    }

    public static void printInfo(Employee empl) {
        System.out.println("salary = " + empl.getSalary());
        System.out.println("days = " + empl.getVacationDays());
        System.out.println("form = " + empl.getVacationForm());
    }
}
```

Output:
salary = 40000.0
vacation days = 15
vacation form = pink

salary = 50000.0
vacation days = 10
vacation form = yellow
Polymorphism and arrays

public class EmployeeMain2 {
    public static void main(String[] args) {
        Employee[] employees = { new Lawyer(), new Secretary(),
                                new Marketer(), new LegalSecretary() };
        for (int i = 0; i < employees.length; i++) {
            System.out.println("salary = " + employees[i].getSalary());
            System.out.println("vacation days = " +
                                employees[i].getVacationDays());
            System.out.println();
        }
    }
}

Output:
salary = 40000.0
vacation days = 15

salary = 40000.0
vacation days = 10

salary = 50000.0
vacation days = 10

salary = 45000.0
vacation days = 10
Assume that the following four classes have been declared:

```java
public class Foo {
    public void method1() {
        System.out.println("foo 1");
    }

    public void method2() {
        System.out.println("foo 2");
    }

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

public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }

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

public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }

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

public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```
Exercise 1

What would be the output of the following client code?

```java
Foo[] pity = { new Baz(), new Bar(),
              new Mumble(), new Foo() };

for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```
Diagramming polymorphic code
public class Foo {
    public void method1() {
        System.out.println("foo 1");
    }

    public void method2() {
        System.out.println("foo 2");
    }

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

public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }
}

public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }

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

public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}

Foo[] pity = { new Baz(),
               new Bar(),
               new Mumble(),
               new Foo() };

for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
Solution 1

- The code produces the following output:

  ```
  baz
  baz 1
  foo 2

  foo
  foo 1
  bar 2

  baz
  baz 1
  mumble 2

  foo
  foo 1
  foo 2
  ```
Exercise 2

- Assume that the following four classes have been declared:

```java
public class Lamb extends Ham {
    public void b() {
        System.out.println("Lamb b");
    }
}

public class Ham {
    public void a() {
        System.out.println("Ham a");
    }
    public void b() {
        System.out.println("Ham b");
    }
    public String toString() {
        return "Ham";
    }
}

public class Spam extends Yam {
    public void a() {
        System.out.println("Spam a");
    }
}

public class Yam extends Lamb {
    public void a() {
        System.out.println("Yam a");
    }
    public String toString() {
        return "Yam";
    }
}
```
Exercise 2

What would be the output of the following client code?

```java
Ham[] food = { new Spam(), new Yam(),
              new Ham(), new Lamb() };

for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println();
}
```
Diagramming polymorphic code
public class Lamb extends Ham {
    public void b() {
        System.out.println("Lamb b");
    }
}

public class Ham {
    public void a() {
        System.out.println("Ham a");
    }
    public void b() {
        System.out.println("Ham b");
    }
    public String toString() {
        return "Ham";
    }
}

Ham[] food = { new Spam(),
                new Yam(),
                new Ham(),
                new Lamb() };

for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println();
}
Solution 2

- The code produces the following output:

```
Yam
Spam  a
Lamb  b

Yam
Yam  a
Lamb  b

Ham
Ham  a
Ham  b

Ham
Ham  a
Lamb  b
```
Variable Shadowing

Something to avoid!!
Variable Shadowing: **Something to avoid!!**

- Polymorphism applies to methods in Java
- **But not to fields!**

```java
public class A {
    int x = 1;
    int method() { return 1; }
}

public class B extends A {
    int x = 2;
    int method() { return 2; }
}
```

```java
A a1 = new A();
A a2 = new B();
System.out.println(a1.method());  // prints 1
System.out.println(a2.method());  // prints 2
System.out.println(a1.x);         // prints 1
System.out.println(a2.x);         // prints 1 still!
```
Variable Shadowing: *Something to avoid!!*

- **Variable Shadowing:**
  - When a class extends another class and defines a field with the same name, each object of the subclass contains *two* fields with that name.
  - The subclass’s version of the field is said to *shadow* the superclass’s version, making the superclass’s version invisible within that class.
  - This is called variable shadowing.
Variable Shadowing: *Something to avoid!!*

Variable Shadowing and References

- If class B extends class A and both have a field of the same name, *references* to objects of type B can access one or the other of the fields.

- The version of the field that they reference depends on the type of the reference variable.
Variable Shadowing: Something to avoid!!

```java
public class A {
    int x = 1;
    int method() { return 1; }
}

public class B extends A {
    int x = 2;
    int method() { return 2; }
}

A a1 = new A();
A a2 = new B();
System.out.println(a1.method()); // prints 1
System.out.println(a2.method()); // prints 2
System.out.println(a1.x); // prints 1
System.out.println(a2.x); // prints 1 still!
// because reference a2 has compile-time type A.
```
Variable Shadowing:
Something to avoid!!

```java
public class A {
    int x = 1;
    int method() { return 1; }
}

public class B extends A {
    int x = 2;
    int method() { return 2; }
}

A a1 = new A();
A a2 = new B();
B b1 = (B)a2;

System.out.println(a1.method()); // prints 1
System.out.println(a2.method()); // prints 2
System.out.println(a1.x); // prints 1
System.out.println(a2.x); // prints 1 still!
System.out.println(b1.x); // prints 2!
// because b1 has static type B
```
## Overriding vs. Variable Shadowing

<table>
<thead>
<tr>
<th>Overriding</th>
<th>Variable Shadowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to methods</td>
<td>Applies to fields</td>
</tr>
<tr>
<td>If subclass overrides a superclass method, it <strong>does not</strong> inherit the superclass method.</td>
<td>If a subclass shadows a superclass field, it <strong>does</strong> inherit the superclass field, but shadows it.</td>
</tr>
<tr>
<td>The behavior of a method call depends on the dynamic (run-time) type of the object.</td>
<td>The behavior of a field access depends on the static (compile-time) type of the reference to the object.</td>
</tr>
</tbody>
</table>
Variable Shadowing: *Something to avoid!!*

- By this time, hopefully you can see that variable shadowing on its own is not that all that complicated, no more than method overriding.

- But if you have to keep track of both method overriding and variable shadowing, then *variable shadowing is very confusing*.

- In general, programmers try to avoid it, and they use method overriding all the time.
Exercise 3

Assume that the following classes have been declared:

```java
public class Ham {
    int a = 0;
    int b = 1;
    public void a() {
        System.out.println("Ham " + a);
    }

    public void b() {
        System.out.println("Ham " + b);
    }

    public String toString() {
        return "Ham " + a + " " + b;
    }
}

public class Spam extends Ham {
    int a = 2;
    public void a() {
        System.out.println("Spam " + a);
    }
}

public class Yam extends Spam {
    int b = 3;
    public void a() {
        System.out.println("Yam " + a);
    }

    public void b() {
        System.out.println("Yam " + b);
    }
}
```
Exercise 3

What would be the output of the following client code?

Ham[] food = { new Spam(), new Yam(),
               new Ham()};

for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println(food[i].a);
    System.out.println(food[i].b);
    System.out.println();
}
```java
public class Ham {
    int a = 0;
    int b = 1;
    public void a() {
        System.out.println("Ham " + a);
    }
    public void b() {
        System.out.println("Ham " + b);
    }
    public String toString() {
        return "Ham " + a + " " + b;
    }
}

public class Spam extends Ham {
    int a = 2;
    public void a() {
        System.out.println("Spam " + a);
    }
}

public class Yam extends Spam {
    int b = 3;
    public void a() {
        System.out.println("Yam " + a);
    }
    public void b() {
        System.out.println("Yam " + b);
    }
}

Ham[] food = { new Spam(),
               new Yam(),
               new Ham() };

for (int i = 0;
     i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println(food[i].a);
    System.out.println(food[i].b);
}
```

Output:

Ham 0 1
Yam 2
Yam 3
0
1