Notes

CSM 61B Spring 2017

Reference & Polymorphism

Week 4: February 6, 2017

1 Switcheroo

1.1 What is wrong with this definition of swap? How can we fix it?

```
class SimpleSwap {
    public static void swap(int a, int b) {
        int temp = b;
        b = a;
        a = temp;
    }
    public static void main(String[] args) {
        int x = 2;
        int y = 5;
        System.out.println("x: " + x + ", y: " + y);
        swap(x, y);
        System.out.println("x: " + x + ", y: " + y);
    }
}
```

```
x: 2, y: 5
x: 2, y: 5
```

In the main method, x and y won't actually be swapped. Within swap, we can change what a and b point to, but we can't change the variables that were declared in main. We can fix this by either in-lining the swap functionality in the main method or returning and reassigning the swapped values using an object.

1.2 How is this implementation of swap different?

```
class Point {
    int x;
    int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
class SwapObject {
    public static void swap(Point p) {
        int temp = p.x;
        p.x = p.y;
        p.y = temp;
    }
    public static void main(String[] args) {
        Point p = new Point(2, 5);
```

```
        System.out.println("p.x: " + p.x + ", p.y: " + p.y);
        swap(p);
        System.out.println("p.x: " + p.x + ", p.y: " + p.y);
    }
}
```

When calling `swap` with a `Point` object, we're passing a reference to the original `Point` object. The object's instance variables can be changed from within `swap` and will remain changed after we exit from the function.

2 Dogs Yay

2.1
```
class Dog {
    public void walk() {
        System.out.println("The dog is walking");
    }
}
class Beagle extends Dog {
    @Override
    public void walk() {
        System.out.println("The beagle is walking");
    }
}
```

What would Java display?

(a)
```
Dog fido = new Dog();
fido.walk();
```

```
The dog is walking
```

(b)
```
Beagle fido = new Beagle();
fido.walk();
```

```
The beagle is walking
```

(c)
```
Beagle fido = new Dog();
fido.walk();
```

Compile-time error.

A container meant for `Beagles` can't contain `Dogs`.

(d)
```
Dog fido = new Beagle();
fido.walk();
```

```
The beagle is walking
```

A container for `Dogs` can contain `Beagles`. At compile time, `fido.walk()` is linked to `Dog.walk()` but at runtime, this method is overridden by `Beagle.walk()`.

2.2
```
class Dog {
    public String className;
    public Dog() {
        className = "dog";
    }
    public String getClassName() {
        return className;
    }
}
class Beagle extends Dog {
    public String className;
    public Beagle() {
        super();
        className = "beagle";
    }
}
class Chihuahua extends Dog {
    public String className;
    public Chihuahua() {
        super();
        className = "chihuahua";
    }
    @Override
    public String getClassName() {
        return className;
    }
}
```

Note that this behavior, *field hiding*, is **not a part of the course**. You will not be expected to know how to solve these problems on an exam.

What would Java display?

(a)
```
Dog d = new Chihuahua();
System.out.println(d.getClassName());
```

chihuahua. `d.getClassName` is an instance method so we look at its dynamic type, `Chihuahua`. `getClassName` returns d's `Chihuahua.className`, chihuahua.

(b)
```
Dog d = new Beagle();
System.out.println(d.className);
```

dog. `d.className` is a field so we use its static type. d's `Dog.className` is dog.

(c)
```
Beagle d = new Beagle();
System.out.println(d.getClassName());
```

dog. `d.getClassName` is an instance method so we use its dynamic type: `Beagle`. `Beagle` inherits `getClassName` from `Dog` so the static type of `this` in `Dog.getClassName` is `Dog`. As a result, we return d's `Dog.className`, dog.

3 Pointy *Extra for Experts*

3.1 What does `mystery` do? *Hint: Draw a box-and-pointer diagram.*

```
public class IntList {
    public int first;
    public IntList rest;
    public IntList(int first, IntList rest) {
        this.first = first;
        this.rest = rest;
    }
    public static IntList mystery(IntList L) {
        if (L == null || L.rest == null) {
            return L;
        } else {
            IntList x = mystery(L.rest);
            L.rest.rest = L;
            L.rest = null;
            return x;
        }
    }
    public static void main(String[] args) {
        IntList x = IntList.list(2, 3, 4, 5);
        System.out.println(x);
        IntList y = mystery(x);
        System.out.println(x);
        System.out.println(y);
    }
}
```

`mystery` reverses the `IntList` in-place and returns the front of the reversed list.