CSM 61B
Spring 2020

Objects & Arrays

Mentoring 4: September 14-19, 2020

1 Switcheroo

The **Golden Rule of Equals** says:

> Given variables `b` and `a`, the assignment statement `b = a`
> copies all the bits from `a` into `b`.

Passing parameters obeys the same rule: copy the bits to the new scope.

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, 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 Coordinate {
    int x, y;
    Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class SwapObject {
    public static void swap(Coordinate p) {
        int temp = p.x;
        p.x = p.y;
        p.y = temp;
    }
    public static void main(String[] args) {
        Coordinate p = new Coordinate(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 `Coordinate` object, we're passing a reference to the original `Coordinate` object. The object's instance variables can be changed from within `swap` and will remain changed after we exit from the function.

2 Flatter Me

Arrays are ordered sequences of fixed length. Unlike Python lists, the `length` must be known when creating an array.

```java
int[] a = new int[3];
```

It is possible to initialize and fill an array in a single expression.

```java
int[] b = new int[]{1, 2, 3};
```

Java can infer the type of the array from its context, yielding this shorthand.

```java
int[] c = {1, 2, 3};
```

Uninitialized values have a default value like `0`, `false`, or `null`.

```java
String[] c = new String[1];
c[0] == null;
```

2.1 Implement `middle`, which takes in `int[]` and returns the middle element. If no element is in the exact middle, return the element to the left middle.

```java
public static int middle(int[] data) {
    return data[(data.length - 1) / 2];
}
```

2.2 Write a method `flatten` that takes in a two-dimensional array `data` and returns a one-dimensional array that contains all of the arrays in `data` concatenated together.

```java
public static int[] flatten(int[][] data) {
    int size = 0;
    for (int[] row : data) {
        size += row.length;
    }
    int[] result = new int[size];
    int i = 0;
    for (int[] row : data) {
        for (int value : row) {
            result[i] = value;
            i += 1;
        }
    }
    return result;
}
```

3 Dogs Yay

3.1
```java
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)
```java
Dog fido1 = new Dog();
fido1.walk();
```

```
The dog is walking
```

(b)
```java
Beagle fido2 = new Beagle();
fido2.walk();
```

```
The beagle is walking
```

(c)
```java
Beagle fido3 = new Dog();
fido3.walk();
```

Compile-time error. A container meant for `Beagles` can't contain `Dogs`.

(d)
```java
Dog fido4 = new Beagle();
fido4.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()`.

3.2 What would each call in `Poodle.main` print? If a line would cause an error, determine if it is a compile-error or runtime-error.

```
class Dog {
    void bark(Dog d) {
        System.out.println("bark");
    }
}

class Poodle extends Dog {
    void bark(Dog d) {
        System.out.println("woof");
    }
    void bark(Poodle p) {
        System.out.println("yap");
    }
    void play(Dog d) {
        System.out.println("no");
    }
    void play(Poodle p) {
        System.out.println("bowwow");
    }
    public static void main(String[] args) {
        Dog dan = new Poodle();
        Poodle pym = new Poodle();

1) dan.play(dan)   // Compile-error     5) pym.bark(dan)   // woof
2) dan.play(pym)   // Compile-error     6) pym.bark(pym)   // yap
3) pym.play(dan)   // no                7) dan.bark(dan)   // woof
4) pym.play(pym)   // bowwow            8) dan.bark(pym)   // woof

}
```

4 Pokemon *Extra Practice*

4.1 Identify the errors that occur when running the code to the right.

```
public class Pokemon {
    public int hp, power;
    public String cry;
    public String secret;
    public Pokemon() {
        hp = 50;
        cry = "Poke?";
    }
    public Pokemon(String c, int hp) {
        cry = c;
        this.hp = hp;
    }
    public void attack(Pokemon p) {
        p.hp -= power;
    }
    public void eat() {
        System.out.println("nom nom");
    }
}
public class Pikachu extends Pokemon {
    public Pikachu() {
        hp = 100;
    }
    public Pikachu(int hp) {
        super("Pika pika pikachu", hp);
    }
    public void attack(Pokemon p) {
        p.hp = 0;
    }
    public void eat() {
        System.out.println("nom Pika nom");
    }
}
public class Squirtle extends Pokemon {
    public void attack() {
        System.out.println("Water gun!!");
    }
}
```

```
Pikachu p = new Pikachu();
Pokemon a = p;
// (1)
// p = a;
a.eat();
a = new Squirtle();
// (2)
// a.attack();
((Squirtle) a).attack();
Pokemon z = new Pikachu();
// (3)
// Squirtle s = (Squirtle) z;
((Pokemon) p).attack(z);
```

`(1)` is a compile-time error since you can't assign a static type `Pokemon` to a variable who has a static type `Pikachu`. Can be solved with a cast.

`(2)` is a compile-time error because `Pokemon` does not have a method with the signature `attack()`.

`(3)` is a run-time error because the dynamic type of `z` (`Pikachu`) cannot be cast to a `Squirtle`.