Java Details

Lecture 08 Summary

- Java Classes
- Instance Variables
- Instance Methods
- Instantiation
- Creation, and Initialization
- Overloading
- Mutability

By the end of this lecture, you will be able to use Java to implement a class model and an object model.

You will also be able to compile and run a Java program (to test these models).

Java Class Definition

```
public class ClassName {

}
```

Instance Variables

```
public class Address {
    private String street;
    private String city;
    private String province;
    private String postalCode;
}
```

- Why is it called an *instance* variable?
- What is stored in memory for each instance of an Address?

Instance Methods

```
public class Customer {
    private String name;
    private Address mailing;

    public void changeAddress(Address newAddress) {
        if (newAddress != null)
            mailing = newAddress;
    }
}
```

What does our block of memory look like after creating the Address and Customer class definitions?

(hint: this is kind of a trick question)

Creating Objects

```
Address anAddress = new Address();
```

- How is `new` different than `malloc`?
- How is it similar?
- What does our block of memory look like after this line of code?
- What does our object model look like?

Initializing Instance Variables

- Method 1: Accessor Methods
  - Setters & Getters
- Method 2: Constructors

Initializing Using Setters

```java
public class Address {
    private String street;
    private String city;
    private String province;
    private String postalCode;

    public String getStreet() {
        return street;
    }

    public void setStreet(String aStreet) {
        street = aStreet;
    }

    public static void main(String[] args) {
        Address anAddress = new Address();
        anAddress.setStreet("123 Any Street");

        Address address2 = new Address();
        address2.setStreet("456 Another Street");

        System.out.println("Street #1: " + anAddress.getStreet());
        System.out.println("Street #2: " + address2.getStreet());
    }
}
```

Compile and Run

```
> javac Address.java
> java Address
```

Exercise: draw the object model and explain the output.

Are there any issues with this setter?

Improved Setter

```
public void setStreet(String aStreet) {
    if (aStreet != null && aStreet.trim().length() > 0)
        street = aStreet;
}
```

What is your reaction to the idea of having to use setters to change the value of instance variables?

What would the code look like if you did this for all of the instance variables?

Initializing Using Constructors

```
public class Address {
    ...

    public Address(String aStreet, String aCity,
                   String aProvince, String aPostalCode) {
        street = aStreet;
        city = aCity;
        province = aProvince;
        postalCode = aPostalCode;
    }

    public static void main(String[] args) {
        Address anAddress =
            new Address("123 Any Street", "Calgary", "AB", "T1T 2T3");
        Address address2 =
            new Address("234 Another Street", "Vancouver", "BC", "V1V 2V3");

        System.out.println("Address #1");
        System.out.println(anAddress.getStreet());
        System.out.println(anAddress.getCity() + ", " + anAddress.getProvince());
        System.out.println(anAddress.getPostalCode());

        System.out.println("Address #2");
        System.out.println(address2.getStreet());
        System.out.println(address2.getCity() + ", " + address2.getProvince());
        System.out.println(address2.getPostalCode());
    }
```

Exercise: draw the object model and explain the output.

Constructors

- Must have the same name as the class

- Cannot have a return type
  - What happens if you add one?

- In our example, what happens if we do this?

```
Address anAddress = new Address();
```

Overloading Constructors

```
public class Address {

    public Address() {
    }

    public Address(String aStreet, String aCity,
            String aProvince, String aPostalCode) {
        street = aStreet;
        city = aCity;
        province = aProvince;
        postalCode = aPostalCode;
    }
}
```

If the constructors have the same name, how does the compiler know which one you’re calling?

Consider the following:

```
public class Vector {
    private float x;

    private float y;

    public Vector(float aX, float aY) {
        x = aX;
        y = aY;
    }

    public Vector(Point a, Point b) {
        x = b.getX() - a.getX();
        y = b.getY() - a.getY();
    }
}
```

How do we decide what constructors to provide?

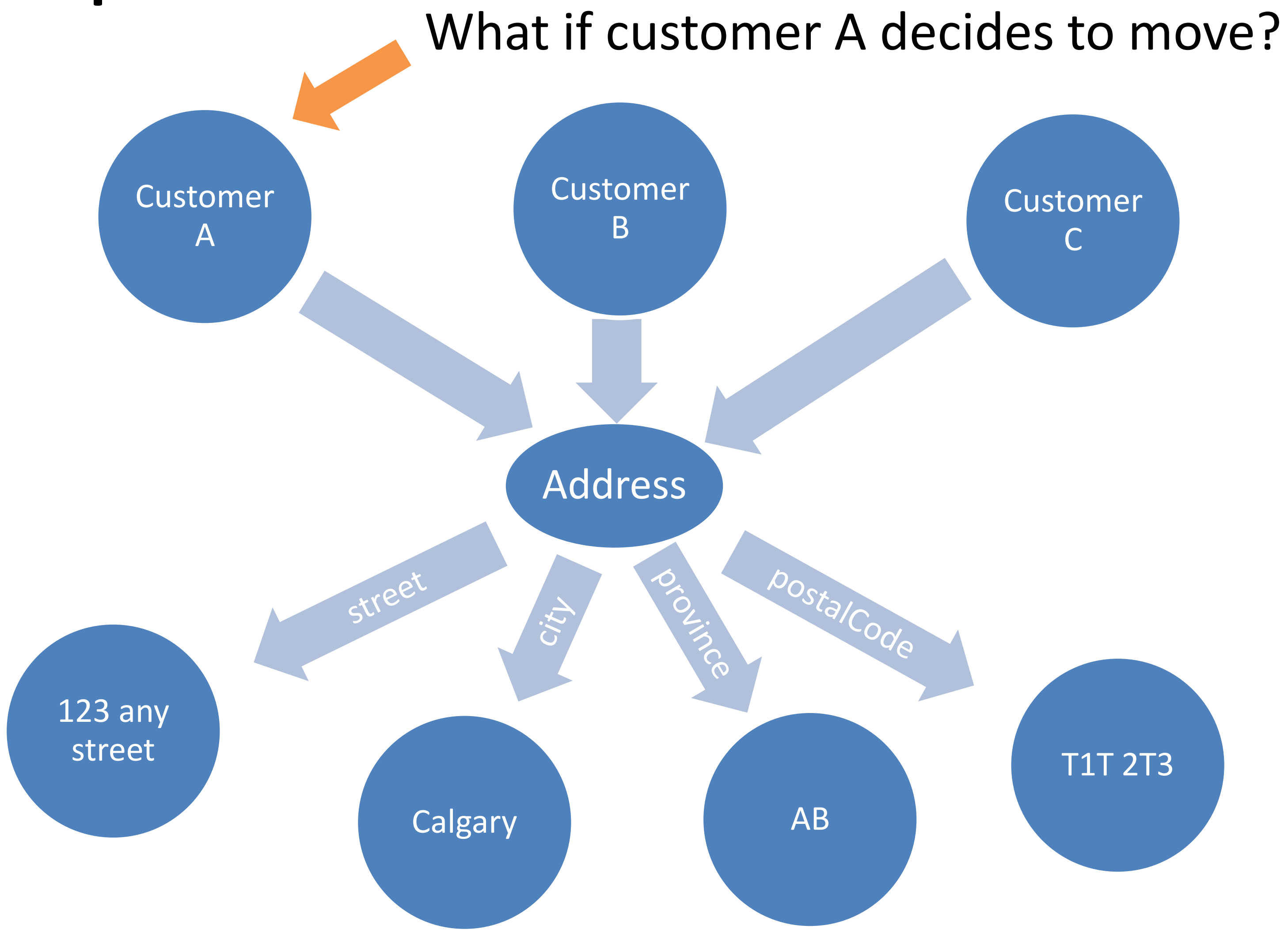
If we provide multiple constructors, an instance variable can be manipulated in many ways.

What is the disadvantage?

How do we fix this problem?

Exercise: decide which constructors might be needed for the Font class we talked about last class and create them.

Example


What if customer A decides to move?
Customer A
Customer B
Customer C
Address
street
city
province
postalCode
123 any street
Calgary
AB
T1T 2T3

Mutability

- Immutable Objects
  - all instance variables *cannot* be changed after creation of the object

Exercise: make our Address class immutable.

Design Decisions

- Should a class be mutable or immutable?
  – no general answer

- Good approach:
  – immutable by default
  – as the program evolves, can do one of:
    - make setters public (class now mutable)
    - make a method to modify instance variables

Is this class immutable?

```
public class Location {
    public float longitude;
    public float latitude;
}
```

Is this class immutable?

```
public class Location {
    private float longitude;
    private float latitude;

    public Location(float lon, float lat) {
        longitude = lon;
        latitude = lat;
    }
}
```

Is this class immutable?

```
public class Location {
    private float longitude;
    private float latitude;

    public Location(float lon, float lat) {
        longitude = lon;
        latitude = lat;
    }

    public void print() {
        System.out.println("Longitude:"+ longitude);
        System.out.println("Latitude:" + latitude);
    }
}
```

Is this class immutable?

```
public class Location {
    private float longitude;
    private float latitude;

    public Location(float lon, float lat) {
        setLongitude(lon);
        latitude = lat;
    }

    public void setLongitude(float lon) {
        longitude = lon;
    }
}
```

Is this class immutable?

```
public class Location {
    private float longitude;
    private float latitude;

    public Location(float lon, float lat) {
        setLongitude(lon);
        latitude = lat;
    }

    private void setLongitude(float lon) {
        longitude = lon;
    }
}
```

Is this class immutable?

```
public class Location {
    private float longitude;
    private float latitude;

    public Location(float lon, float lat) {
        longitude = lon;
        latitude = lat;
    }

    public void add(float lon, float lat) {
        longitude += lon;
        latitude += lat;
    }
}
```

Is this class immutable?

```
public class Customer {
    private String name;
    private Address address;

    public Address(String n, Address a) {
        name = n;
        address = a;
    }

    public Address getAddress() {
        return address;
    }
}
```

Lecture 08 Summary

- Java Classes
- Instance Variables
- Instance Methods
- Instantiation
- Creation, and Initialization
- Overloading
- Mutability

Next Class

- Navigability
- Multiplicity
- Class Variables and Methods