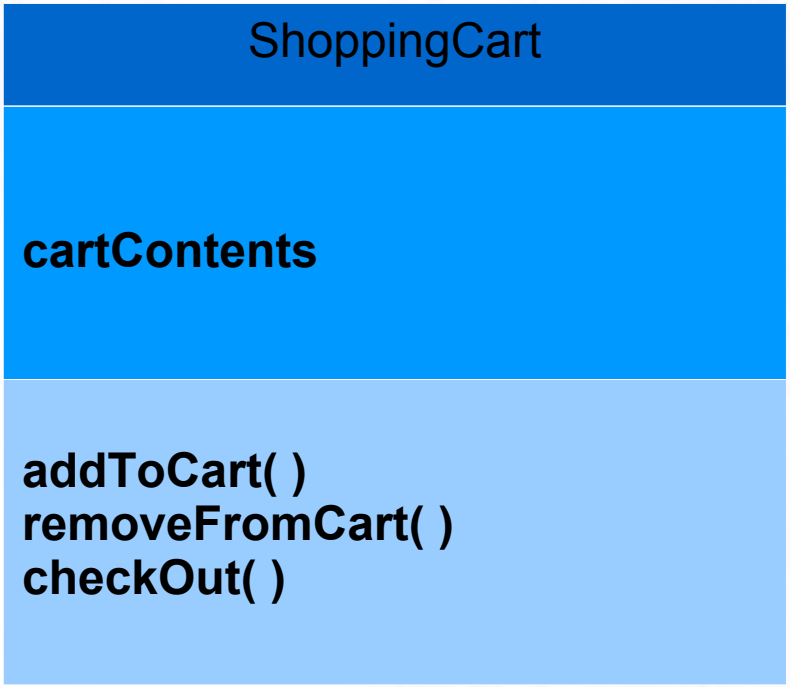
Java Review – Classes and Objects

- When you design a class, think about the objects that will be created from that class type. Think about:
  - Things the object **knows**
  - Things the object **does**

Thinking about Objects



knows

does

Thinking about Objects

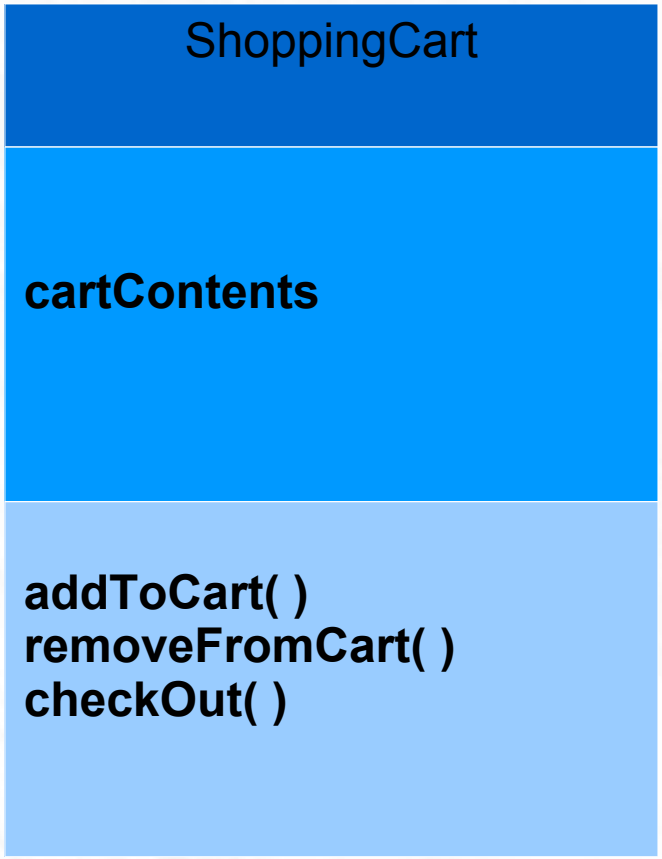
- Things an object knows about itself are called:
  - Instance variables

- Things an object can do are called:
  - Methods

Thinking about Objects

Instance Variables
(state)

Methods
(behavior)



knows

does

Thinking about Objects

- Things an object ***knows*** are called **instance variables**.
- They represent an object's state (the data), and can have unique values for each object.

Thinking about Objects

- Things an object can ***do*** are called **methods**.
- You design the methods to operate on the data (instance variables) within the class.
- It's common for an object to have methods that read or write the values of the instance variables.

Thinking about Objects

- Think of **instance** as another way of saying **object**.

- Objects have instance variables and methods, but those instance methods are designed as part of the class.

What's the difference between a Class and an Object?

- A class is **not** an object (but it's used to construct them).
- **A class is a *blueprint* for an object**.
- It tells the JVM *how* to make an object of that particular type.
- Each object made from that class can have it's own values for instance variables of that class.

Classes and Objects

One class..

dog
size breed name
bark( )

Many objects.

Look at it this way..

- An object is like one entry in your address book.

Name Bryan Dulaney

Phone 248-214-1120

Email bryan.c.dulaney@gmail.com

- Each card has the same blank fields (instance variables)
- When you fill out a card you are creating an instance (object), and the entries you make on the card represent its state.
- The methods of the class are the things you do to a particular card; getName(), changeName(), setName() could all be methods for your address book.
- Each card can *do* the same things (via methods), but each card *knows* things unique to that particular card.

Making your first Object

- What does it take to creat and use an object?
- You need *two* classes:
- One class for the type of object you want to use
- One class to *test* your new class.

Making your first Object

- The *test* class is where you put your main method
- In the main( ) method you create and access objects of your new class type.
- The test class has only one job: to *try out* the methods and variables of your new object class type.
- Use the dot operator (.) to access the methods and variables of the new objects.

Making your first Object

- The Dot Operator (.)

- The dot operator (.) gives you access to an object's state and behavior (instance variables and methods).

```
//make a new object
Dog d = new Dog( ) ;


// tell it to bark using the dot operator on the variable d to call bark( )
d.bark( );
//set its size using the dot operator
d.size = 40;
```

Making your first Object

1. **Write your class**

```
class Dog {
        int size;
        String breed;
        String name;

        void bark( ) {
                System.out.println(“Ruff! Ruff!”);
        }
}
```

Making your first Object

2. Write a test class

```
class DogTest {
        public static void main (String [ ] args) {
                //Dog test code goes here
        }
}
```

Making your first Object

3. In your test class, make an object and access the object's variables and methods.

```
class DogTest {
        public static void main (String [ ] args) {
                Dog d = new Dog( ); //make a Dog object
                d.size = 40;  // use the dot operator to set the size of the Dog
                d.bark( );  // call the Dog's bark( ) method.
        }
}
```

Get out of main( ) !

- As long as you're in main( ) you're not really OO programming. It's fine for a test program to run within the main method, but a true OO application needs object talking to other objects, as opposed to a static main( ) method creating and testing objects.

Get out of main( ) !

The two uses of main:

1. to **test** your real class
2. to **launch/start** your Java **application**

- A real Java application is nothing but objects talking to other objects. In this case, *talking* means objects calling methods on one another.

Bullet Points

- Object-orientd programming lets you extend a program without having to touch previously-tested, working code.
- All Java code is defined in a **class**.
- A class describes how to make an object of that class type. **A class is like a blueprint**.
- An object can take care of itself; you don't have to know or care *how* the object does it.
- An object **knows** things and **does** things.

Bullet Points

- Things an object knows about itself are called **instance variables**. They represent the *state* of an object.
- Things an object does are called **methods**. They represent the *behavior* of an object.
- When you create a class, you may also want to create a separate test class which you'll use to create objects of your new class type.

Bullet Points

- A class can **inherit** instance variables and methods from a more abstract **superclass**.
- At runtime, a Java program is nothing more than objects 'talking' to other objects.

Excercise