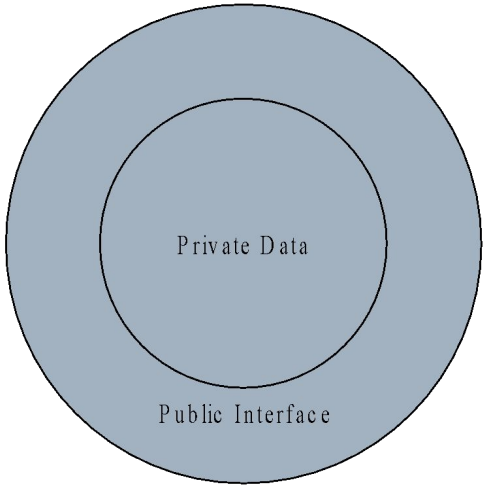
Java –
Introduction to Class

Properties of OOP

Encapsulation

- The data (state) of an object is private – it cannot be accessed directly.
- The state can only be changed through its public *interface*.
- This is called *encapsulation*

"The Doughnut Diagram" Showing that an object has private state and public behaviour. State can only be changed by invoking some behaviour



Classes

Preparation

- Scene so far has been background material and experience
  - Variables
  - Data Types
  - Input and output
  - Expressions
  - Assignments
  - Objects
  - Standard classes and methods
  - Decisions (if, switch)
  - Loops (while, for, do-while)

- Now: Experience what Java is really about
  - Design and implement objects representing information and physical world objects

Object-oriented programming

- Basis
  - Create and manipulate objects with attributes and methods that the programmer can specify
- Mechanism
  - Classes
- Benefits
  - An information type is designed and implemented once
    - Reused as needed
      - No need reanalysis and re-justification of the representation

Known Classes

- Classes we've seen
  - String
  - Scanner
  - System

The Car class

A new example: creating a Car class

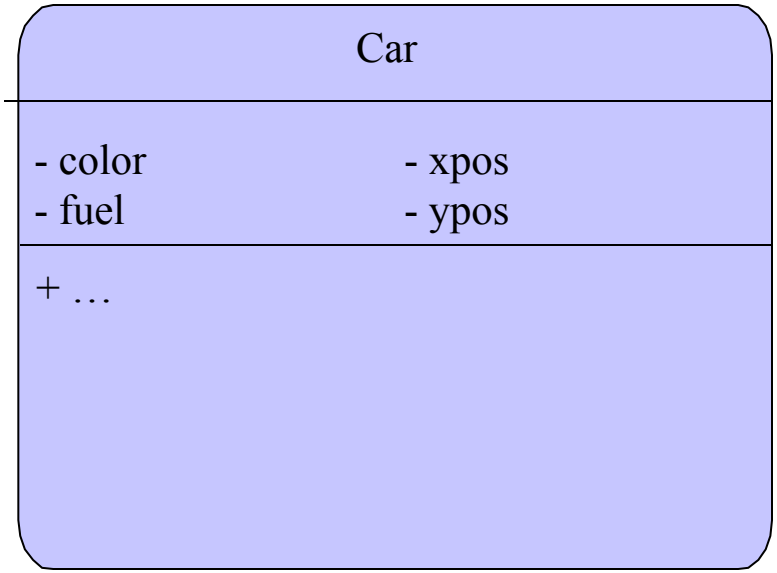
- What properties does a car have in the real world?
  - Color
  - Position (x,y)
  - Fuel in tank

- We will implement these properties in our Car class

```
class Car {
    private Color color;
    private int xpos;
    private int ypos;
    private int fuel;

    //...
}
```

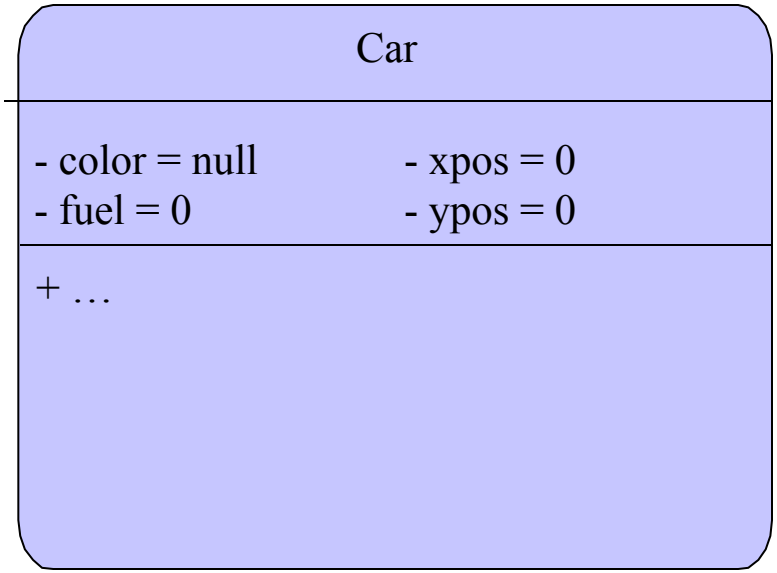
Car's instance variables

```
class Car {
    private Color color;
    private int xpos;
    private int ypos;
    private int fuel;

    //...
}
```



Instance variables and attributes

- Default initialization
  - If the variable is within a method, Java does NOT initialize it
  - If the variable is within a class, Java initializes it as follows:
    - Numeric instance variables initialized to 0
    - Logical instance variables initialized to false
    - Object instance variables initialized to null



Car behaviors or methods

- What can a car do? And what can you do to a car?
  - Move it
    - Change it's x and y positions
  - Change it's color
  - Fill it up with fuel

- For our computer simulation, what else do we want the Car class to do?
  - Create a new Car
  - Change Car's condition
- Each of these behaviors will be written as a method

Creating a new car

- To create a new Car, we call:
  - Car car = new Car();
- Notice this looks like a method
  - You are calling a special method called a constructor
  - A constructor is used to create (or construct) an object
    - It sets the instance variables to initial values
- The constructor:

```
Car() {
fuel = 1000;
color = Color.BLUE;
}
```

Constructors

No return type!

EXACT same name as class

```
Car() {
    fuel = 1000;
    color = Color.BLUE;
}
```

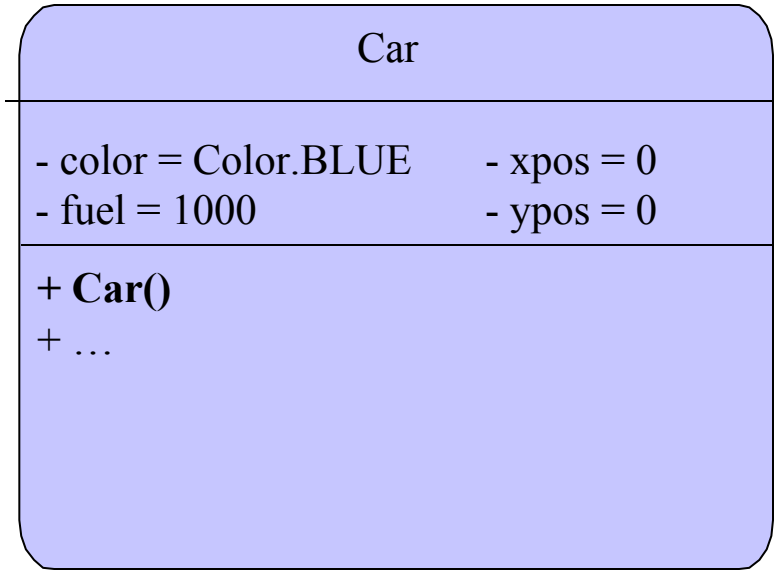
Our Car class so far

```
class Car {
   private Color color;
   private int xpos;
   private int ypos;
   private int fuel;

      Car() {
            fuel = 1000;
            color = Color.BLUE;
       }
}
```

```
class Car {
   private Color color =
                  Color.BLUE;
   private int xpos;
   private int ypos;
   private int fuel = 1000;

      Car() {
      }
}
```

Our Car class so far

```
class Car {
    private Color color =
                    Color.BLUE;
    private int xpos = 0;
    private int ypos = 0;
    private int fuel = 1000;

        Car() {
        }
}
```



- Called the default constructor
  - The default constructor has no parameters
  - If you don't include one, Java will SOMETIMES put one there automatically

Another constructor

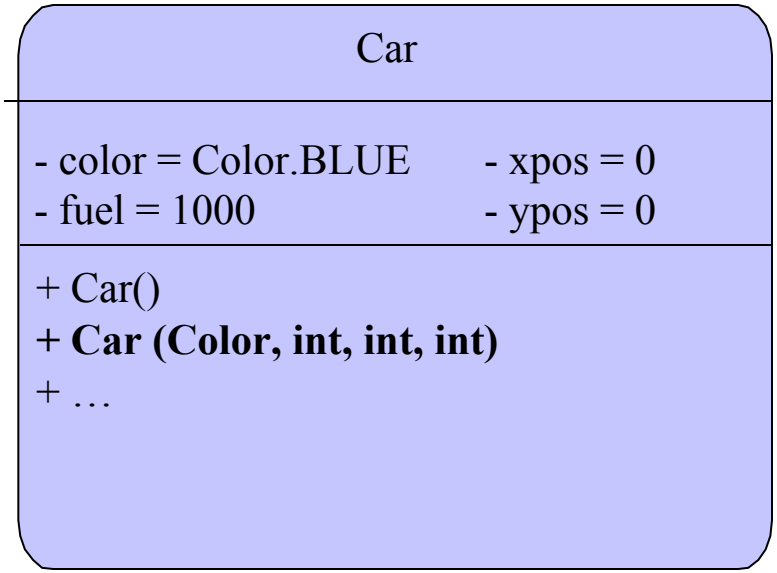
- Another constructor:

```
Car (Color c, int x, int y, int f) {
    color = c;
    xpos = x;
    ypos = y;
    fuel = f;
}
```

- This constructor takes in four parameters
- The instance variables in the object are set to those parameters
- This is called a specific constructor
  - An constructor you provide that takes in parameters is called a specific constructor

Our Car class so far

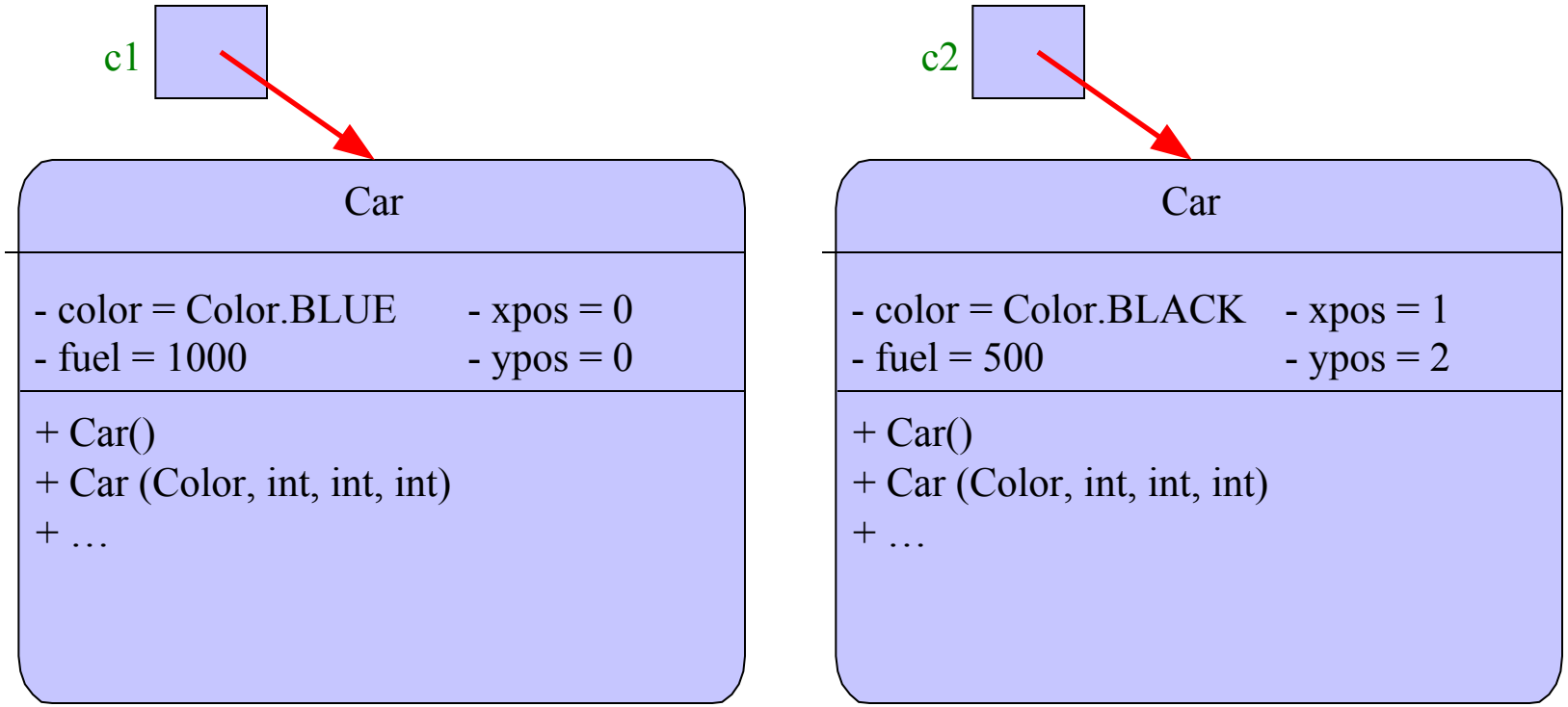
```
class Car {
   private Color color =
            Color.BLUE;
   private int xpos = 0;
   private int ypos = 0;
   private int fuel = 1000;

      Car() {
      }

      Car (Color c, int x, int y, int f) {
            color = c;
            xpos = x;
            ypos = y;
            fuel = f;
      }

}
```



Using our Car class

- Now we can use both our constructors:

```
Car c1 = new Car();
Car c2 = new Car (Color.BLACK, 1, 2, 500);
```



So what does private mean?

- Consider the following code

Note that it's a different class!

```
class CarSimulation {
    public static void main (String[] args) {
        Car c = new Car();
        System.out.println (c.fuel);
    }
}
```

- Recall that fuel is a private instance variable in the Car class
- Private means that code outside the class CANNOT access the variable
  - For either reading or writing
- Java will not compile the above code
  - If fuel were public, the above code would work

So how do we get the fuel of a Car?

- Via accessor methods in the Car class:

```
public int getFuel() {
      return fuel;
}

public Color getColor() {
   return color;
}
```

```
public int getYPos() {
   return ypos;
}

public int getXPos() {
            return xpos;
      }
```

- As these methods are within the Car class, they can read the private instance variables
- As the methods are public, anybody can call them

So how do we set the fuel of a Car?

- Via mutator methods in the Car class:

```
public void setFuel (int f) {
        fuel = f;
}

public void setColor (Color c) {
        color = c;
}
```

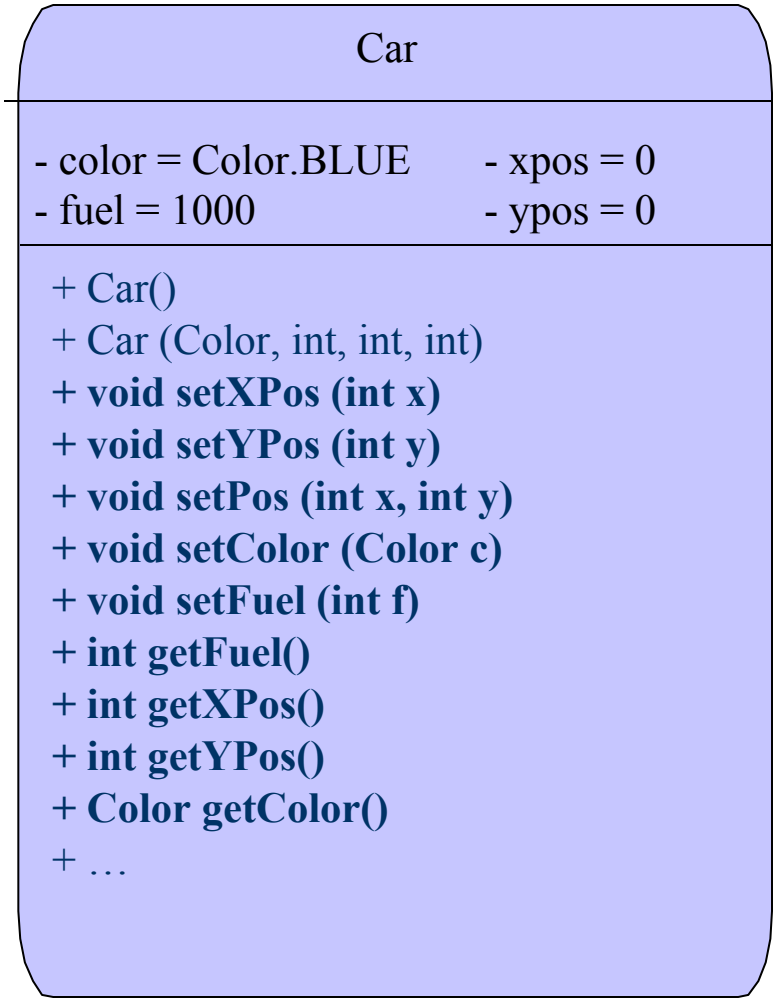
```
public void setXPos (int x) {
   xpos = x;
}

public void setYPos (int y) {
   ypos = y;
}
```

- As these methods are within the Car class, they can read the private instance variables
- As the methods are public, anybody can call them

Why use all this?

- These methods are called a get/set pair
  - Used with private variables

- Our Car so far:



Back to our specific constructor

```
class Car {
    private Color color =
            Color.BLUE;
    private int xpos = 0;
    private int ypos = 0;
    private int fuel = 1000;

    Car (Color c,
        int x, int y, int f) {
        color = c;
        xpos = x;
        ypos = y;
        fuel = f;
  }
}
```

```
class Car {
    private Color color =
            Color.BLUE;
    private int xpos = 0;
    private int ypos = 0;
    private int fuel = 1000;

    Car (Color c,
        int x, int y, int f) {
        setColor (c);
        setXPos (x);
        setYPos (y);
        setFuel (f);
  }
}
```

Back to our specific constructor

- Using the mutator methods (i.e. the ‘set’ methods) is the preferred way to modify instance variables in a constructor

So what's left to add to our Car class?

- What else we should add:
  - A mutator that sets both the x and y positions at the same time
  - A means to "use" the Car's fuel

- Let's do the first:

```
public void setPos (int x, int y) {
setXPos (x);
setYPos (y);
}
```

- Notice that it calls the mutator methods

Using the Car's fuel

- Whenever the Car moves, it should burn some of the fuel
  - For each pixel it moves, it uses one unit of fuel
    - We could make this more realistic, but this is simpler

```
public void setXPos (int x) {
        xpos = x;
}

public void setYPos (int y) {
        ypos = y;
}
```

```
public void setXPos (int x) {
        fuel -= Math.abs
                        (getXPos()-x);
        xpos = x;
}

public void setYPos (int y) {
        fuel -= Math.abs
                        (getYPos()-y);
        ypos = y;
}
```

Using the Car's fuel

```
public void setPos (int x, int y) {
    setXPos(x);
    setYPos(y);
}
```

- Notice that to access the instance variables, the accessor methods are used
- Math.abs() gets the absolute value of the passed parameter

The main() method

- Consider a class with many methods:

```
public class WhereToStart {
    public static void foo (int x) {
        // ...
    }
    public static void bar () {
        // ...
    }
    public static void main (String[] args) {
        // ...
    }
}
```

- Where does Java start executing the program?
  - Always at the beginning of the main() method!

Running a class without a main() method

- Consider the Car class
  - It had no main() method!
  - Create another class named “CarSimulation” where main function and Car class is declared.

- So let’s try running it…