Principles of Programming

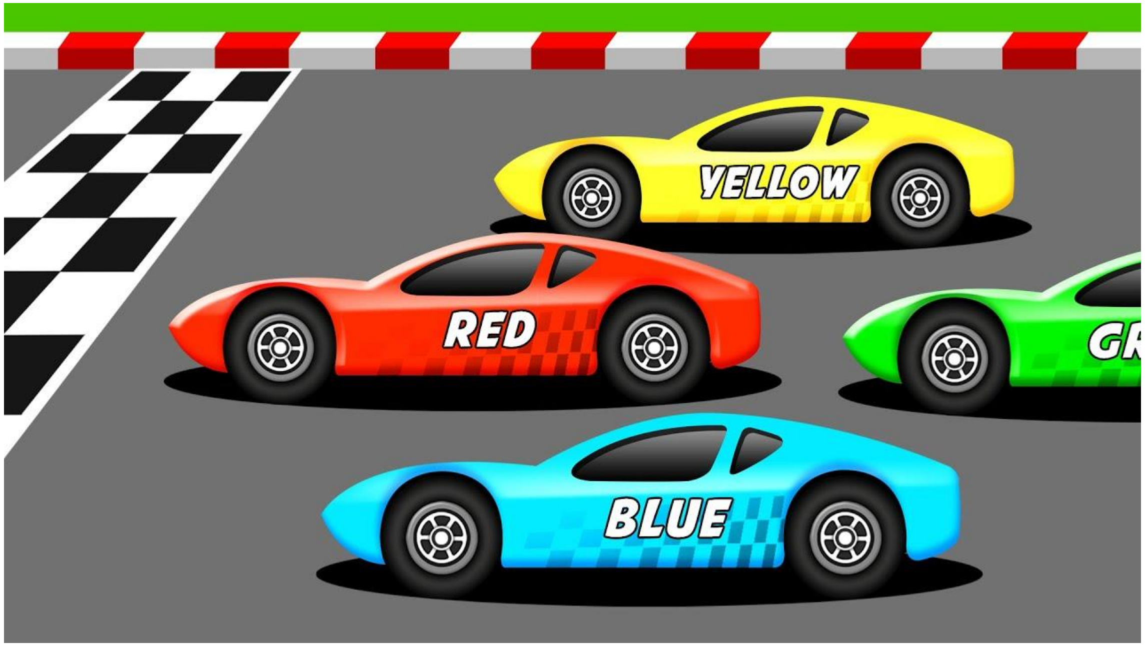
Module 8: Objects and Classes

Object Oriented Programming, Implementing a Simple Class

Today’s Objectives

- Topics
  - 8.1 Object Oriented Programming
  - 8.2 Implementing a Simple Class

A Car Race!



CaesarCipher.java

Last Lecture *on a Page*

- Java programs can use command line arguments.

```
public static void main(String[] args) { ... }
```

```
>java CaesarCipher -d encrypt.txt output.txt
```

args[0] args[1] args[2]

- "Dangerous" code should be in a `try` block:

Handled

```
try {
   String filename = "input.txt";
   Scanner in = new Scanner(new File(filename));
}
catch (FileNotFoundException f) {
   System.out.println("File not found");
}
```

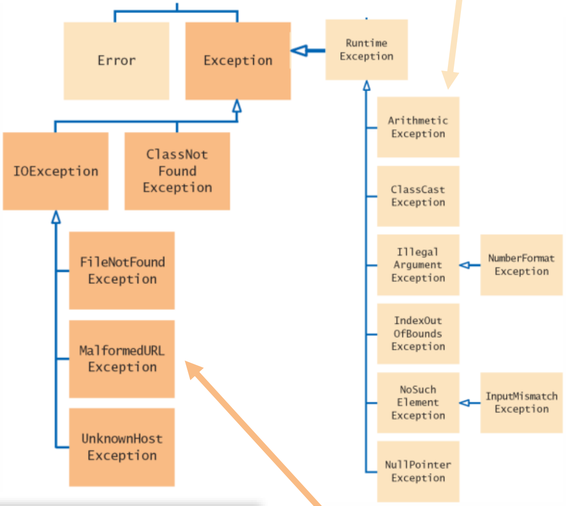
!

- Or declare that you might `throw` the Exception:

Declared

```
public void readStuff(String filename) throws FileNotFoundException {
   ...
}
```

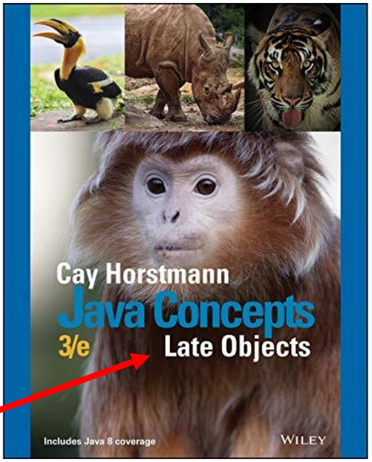
Java Exceptions



"Unchecked Exceptions" do not.

"Checked Exceptions" must be handled or declared.

Object-Oriented Programming



- You have learned structured programming:
  - Breaking tasks into subtasks.
  - Writing reusable methods to handle tasks.
- We will now study object-oriented programming:
  - Using Objects and Classes to build larger and more complex programs.
  - Also used to model objects we use in the world.

A **class** describes a set of objects with the same information and the same behavior.

- The *class* `Car` describes *all* passenger vehicles.
- The *class* `String` describes *all* character strings.

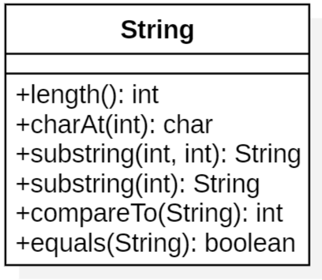
An **object** is a member of a class.

- I own a particular `Car` *object*.
- `Suspect` is a particular `String`.

```
String suspect = "Prof. Plum";
```

Objects and Programs

- Java programs are made of objects that interact with each other.
  - Each object is is a member of a particular class.
  - That class defines the data for the objects and its behavior.
- Behavior: the methods to use with its objects
  - For example, the `String` class provides methods:
    - Examples: `length()` and `charAt()` methods


String

+length(): int
+charAt(int): char
+substring(int, int): String
+substring(int): String
+compareTo(String): int
+equals(String): boolean

class `String`

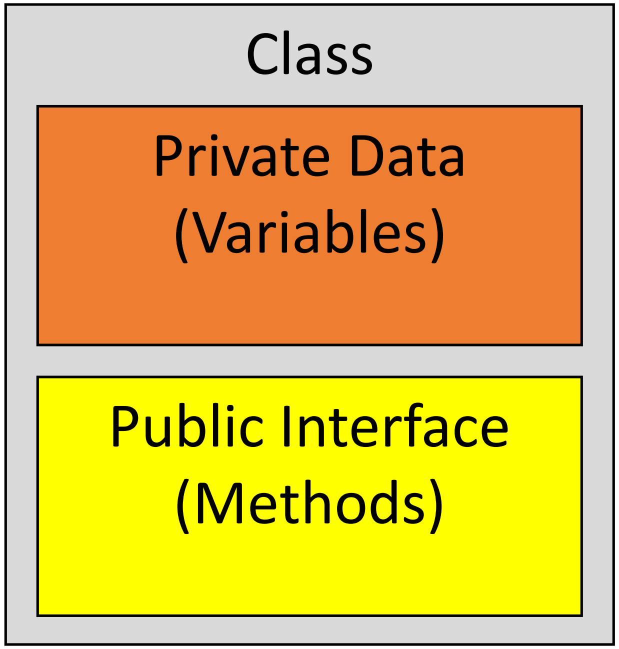
object `suspect`

```
String suspect = "Prof. Plum";
int len = suspect.length();
char c1 = suspect.charAt(0);
```

Calling a *method*, defined by *class* `String`, on *object* `suspect`

Diagram of a Class

- Private Data
  - Each object has its own private data that other objects cannot directly access.
- Public Interface
  - Each object has a set of methods available for other objects to use.
- This is called *Encapsulation*.

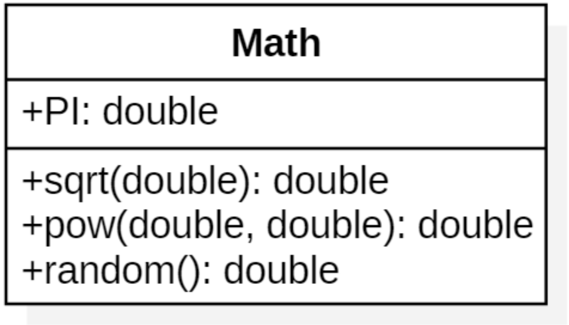


Class

Private Data
(Variables)

Public Interface
(Methods)

Notice the subtle introduction to UML notation for classes!



Math

+PI: double

+sqrt(double): double
+pow(double, double): double
+random(): double

Why we use Encapsulation

- Programmer using a class -
  - Only needs to know what the class does,
  - Not how it does it.

- Programmer writing a class -
  - Can protect the data needed to ensure correct behavior.
  - Can change the implementation later without effecting others.

- Manage Complexity!

Classes and Objects

- A class is a template definition of the methods and variables common for a particular set of objects.
- An object is a specific instance of a class; it contains real values instead of variables.
- A class is the cookie cutter, an object is the cookie.

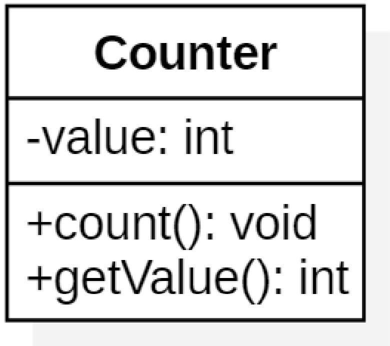
Implementing a Simple Class

- Tally Counter: A class that models a mechanical device that is used to count people.
  - For example, to find out how many people attend a concert or board a bus.

- What does it need to keep track of?
  - The current count

- What should it do?
  - Increment the tally
  - Get the current total

Class diagram in UML

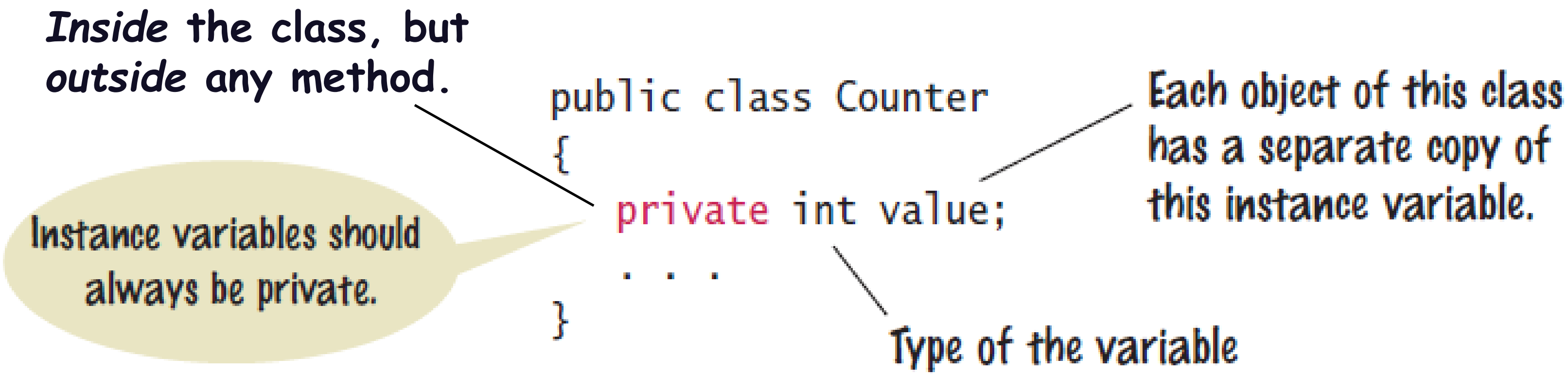

Counter

-value: int

+count(): void
+getValue(): int

Tally Counter Class

- Specify instance variables in the class declaration:



```
public class Counter
{
   private int value;
   . . .
}
```

Inside the class, but outside any method.

Instance variables should always be private.

Each object of this class has a separate copy of this instance variable.

Type of the variable

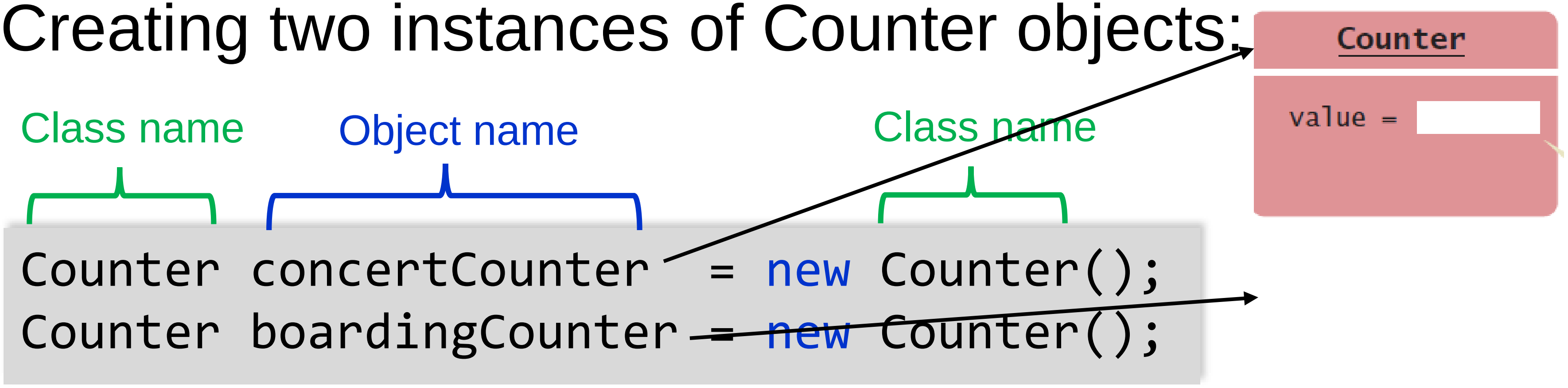
- Each object instantiated from the class has its own set of instance variables.
  - Each tally counter has its own current count (`value`).
- Access Specifiers:
  - Classes are `public`
  - Instance variables are almost always `private`
  - Most methods are `public`

Instantiating Objects

- Objects are created based on classes.
  - Use the new operator to construct objects.
  - Give each object a unique name (like variables).
- You have used the new operator before:

```
Scanner in = new Scanner(System.in);
```

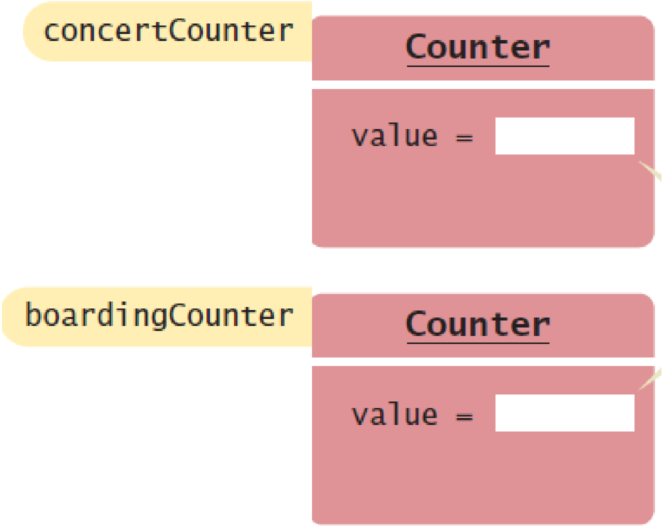
- Creating two instances of Counter objects:



```
Counter concertCounter  = new Counter();
Counter boardingCounter = new Counter();
```

Tally Counter Methods

- Design a method named `count` that adds 1 to the instance variable.
- Which instance variable?
  - Use the name of the object:
    - `concertCounter.count()`
    - `boardingCounter.count()`



concertCounter

Counter

value =

boardingCounter

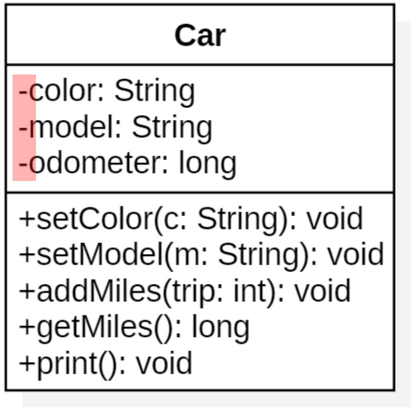
Counter

value =

```
public class Counter
{
   private int value = 0;

   public void count()
   {
      value = value + 1;
   }

   public int getValue()
   {
      return value;
   }
}
```

Another Class Example

- Define a Car class.
- What instance variables might it have?
- What methods will it have?



Car

-color: String
-model: String
-odometer: long

+setColor(c: String): void
+setModel(m: String): void
+addMiles(trip: int): void
+getMiles(): long
+print(): void

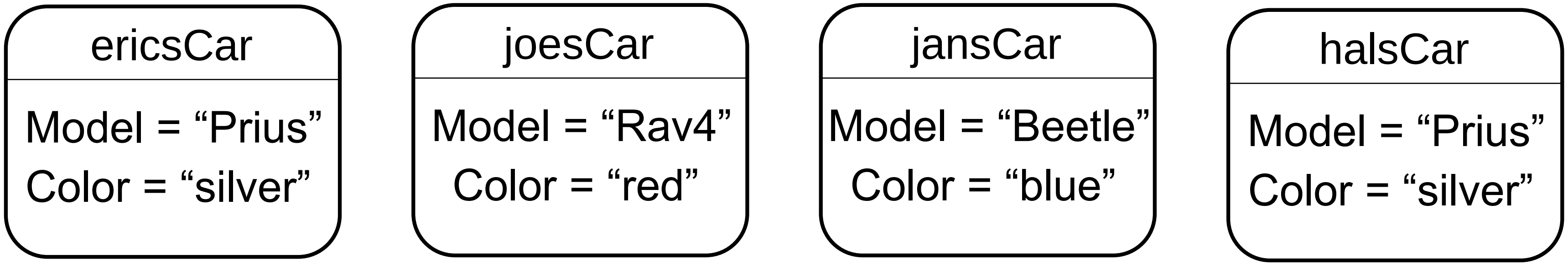
```
class Car {
   private String color;
   private String model;
   private long odometer;

   public void setColor(String c) {
      ...
   }
   ...
}
```

Typically private

'*global*'
Declared *inside* the class, but *outside* any method.

Objects

- A *running* Java program is made up of objects.
- Each object has values for its instance variables.



ericsCar
Model = “Prius”
Color = “silver”

joesCar
Model = “Rav4”
Color = “red”

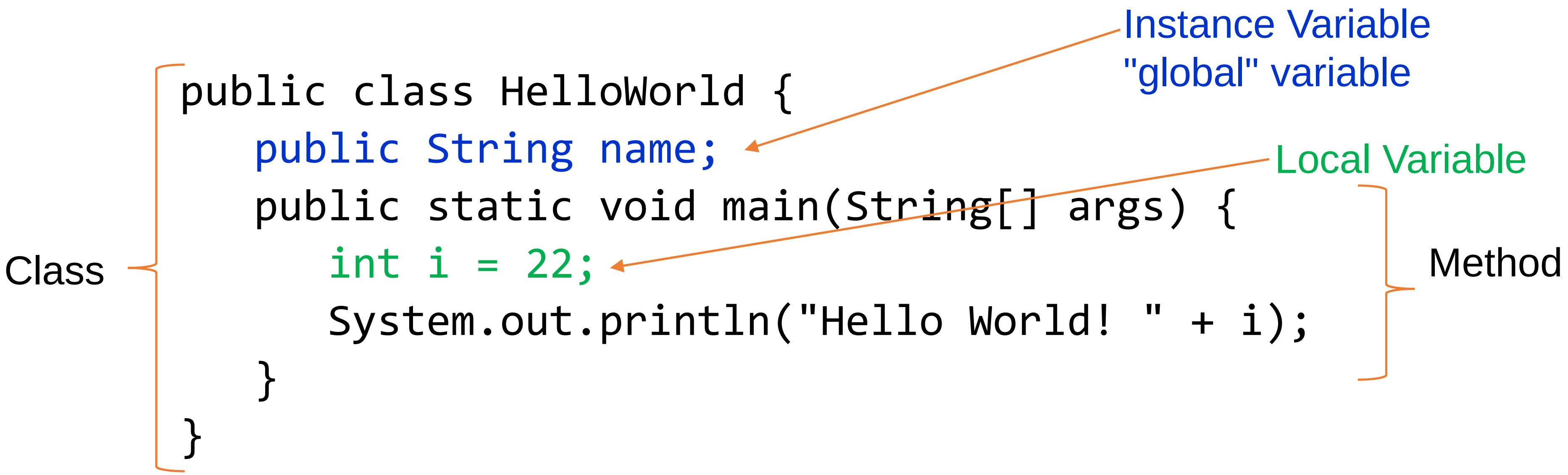
jansCar
Model = “Beetle”
Color = “blue”

halsCar
Model = “Prius”
Color = “silver”

- Objects are created in Java with the *new* statement

```java
Car ericsCar = new Car();
ericsCar.setColor("silver");
ericsCar.setModel("Prius");

Car joesCar = new Car();
joesCar.setColor("red");
joesCar.setModel("Rav 4");
```

Local vs. 'Global' (instance) Variables



```
public class HelloWorld {
    public String name;
    public static void main(String[] args) {
        int i = 22;
        System.out.println("Hello World! " + i);
    }
}
```

Instance Variable
"global" variable

Local Variable

Class

Method

Using Instance Variables

- The code in the methods can use the instance attributes.
  - A method is called on a particular object.
  - The method will use the instance variables for that object.
- Example:

```
Class Car {
  private String color;
  private long odometer = 0;

  public void setColor(String c) {
     color = c;
  }
  public void addMiles(int tripMileage) {
     odometer = odometer + tripMileage;
  }
}
```

Using Classes

- Example:

```
class CarTest {
   public static void main(String[] args) {
      Car c1 = new Car();
      Car c2 = new Car();

      c1.setColor("white");
      c1.addMileage(1200);
      c1.addMileage(100);

      c2.setColor("blue");
   }
}
```

- Result:
  - c1: white, 1300 miles
  - c2: blue: 0 miles

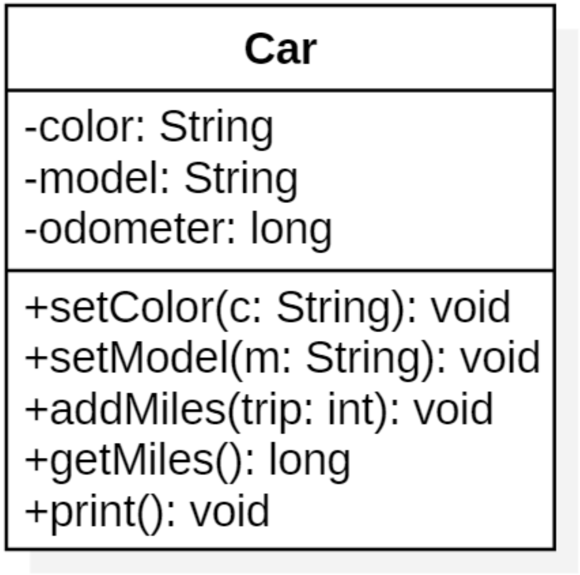
Things to Remember

- Each class must be in its own file.
  - As always, the name of the class must match the name of the file.

- To use another class
  - It must be in the same package
  - Or, you must use the import

Car.java
CarRace.java

```
public class Car {
   private String color;
   private String model;
   private long odometer = 0;

   public void setColor(String c) {
      color = c;
   }

   public void setModel(String m) {
      model = m;
   }

   public void addMiles(int tripMileage) {
      odometer = odometer + tripMileage;
   }

   public long getMiles() {
      return odometer;
   }

   public void print() {
      System.out.println("At " + odometer + ", the " + color + " " + model);
   }
}
```



Car

-color: String
-model: String
-odometer: long

+setColor(c: String): void
+setModel(m: String): void
+addMiles(trip: int): void
+getMiles(): long
+print(): void

CSE110 – Principles of Programming

Eric Eckert

eric.eckert@asu.edu

Disclaimer: These slides can only be used as study material for the class CSE110 at ASU. They cannot be distributed or used for another purpose.