

COMP 110/L Lecture 7

Mahdi Ebrahimi

Some slides adapted from Dr. Kyle Dewey

Outline

- Introduction to objects
 - Constructors and `new`
 - Instance variables
 - Instance methods
 - `static` vs. `non-static`

Object-Oriented Programming

Example

Class: Human

Object: Man, Woman, Child

Class: Fruit

Object: Apple, Banana, Mango

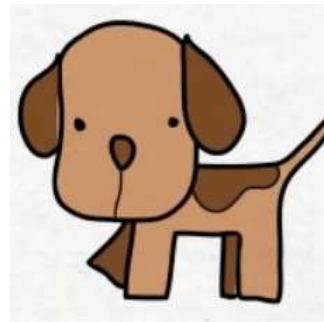
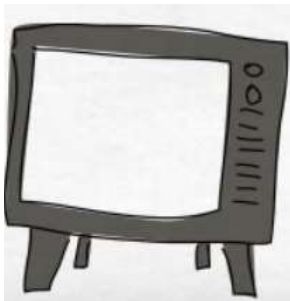
Class: Mobile Phone

Object: iPhone, X Samsung S10

Class: Food

Object: Pizza, Burger, Rice

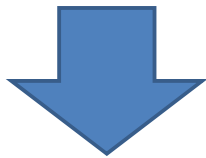
What is an Object?



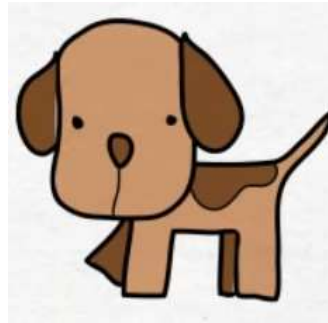
Real-world objects has 2 characteristics

1 - State

Name
Color
Breed

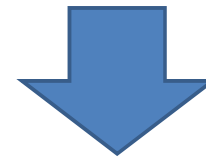


Variables (Fields)



2 - Behavior

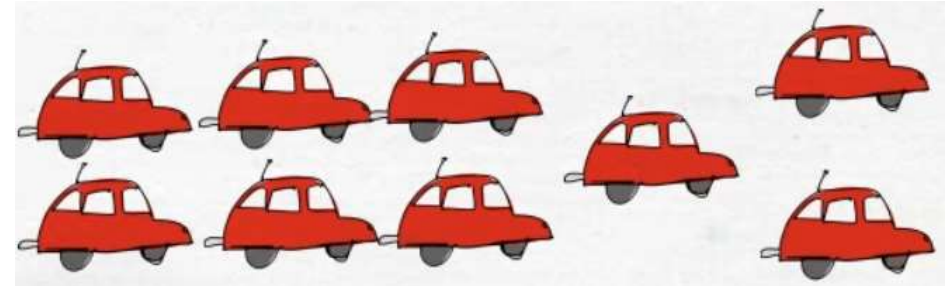
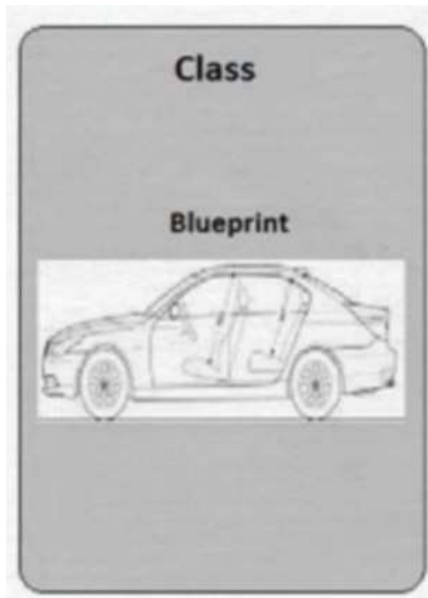
barking
wagging tail



Methods (Functions)

What is Class?

A class is the **blueprint** which individual objects are created.



Blueprint or **template** ?

objects i.e. instance of class (blueprint/template/structure/design)

In real-world car is an object and will have 2 characteristics.



1 - State

Size
Color
Make
Model

Variables (Fields)

2 - Behavior

Move
Accelerate
Turn
Reverse
Shift

Methods (Functions)

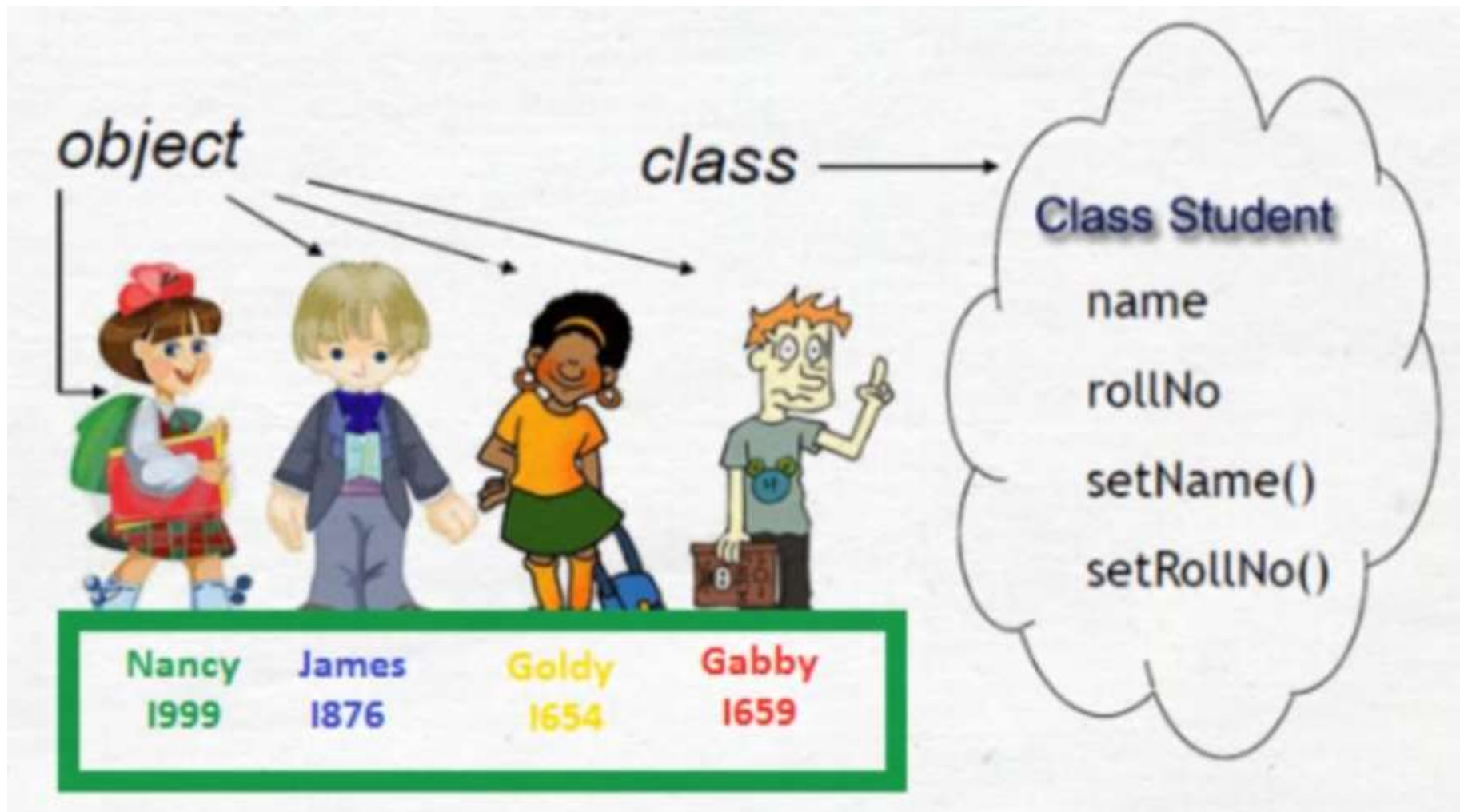
In Programming

```
public class Car {  
    int size; Variables (state)  
    String color;  
    Methods (behavior)  
    public void setSize(int s){  
        this.size = s;  
    }  
  
    public void setColor(String c){  
        this.color = c;  
    }  
    //more code goes here  
}
```

```
public static void main(String[] args) {  
  
    Car carObject1 = new Car();  
    Car carObject2 = new Car();  
    Car carObject3 = new Car();  
  
    carObject1.setSize(6);  
    carObject1.setColor("blue");  
  
    carObject1.setSize(4);  
    carObject1.setColor("red");  
  
    carObject1.setSize(8);  
    carObject1.setColor("grey");  
}
```



Example 2



Basic Idea

The world is composed of *objects*
which **interact** with each other in well-defined ways

Basic Idea

The world is composed of *objects*
which interact with each other in well-defined ways

Example: boiling water

Basic Idea

The world is composed of *objects*
which interact with each other in well-defined ways

Example: boiling water



faucet object

Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water



faucet object



pot object

Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water



faucet object

Interaction:
fill with water



pot object

Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water



faucet object

Interaction:
fill with water



pot object

Basic Idea

The world is composed of *objects*
which interact with each other in well-defined ways

Example: boiling water



pot object

Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water

Interaction:

Place on top of



stove object



pot object

Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water



stove object

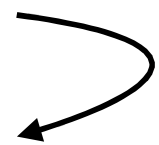
Basic Idea

The world is composed of *objects* which interact with each other in well-defined ways

Example: boiling water



stove object



Interaction:
Turn on burner

Creating Objects

In Java, we first need a *class* to make an *object*.
A class serves as a blueprint/template for an object.

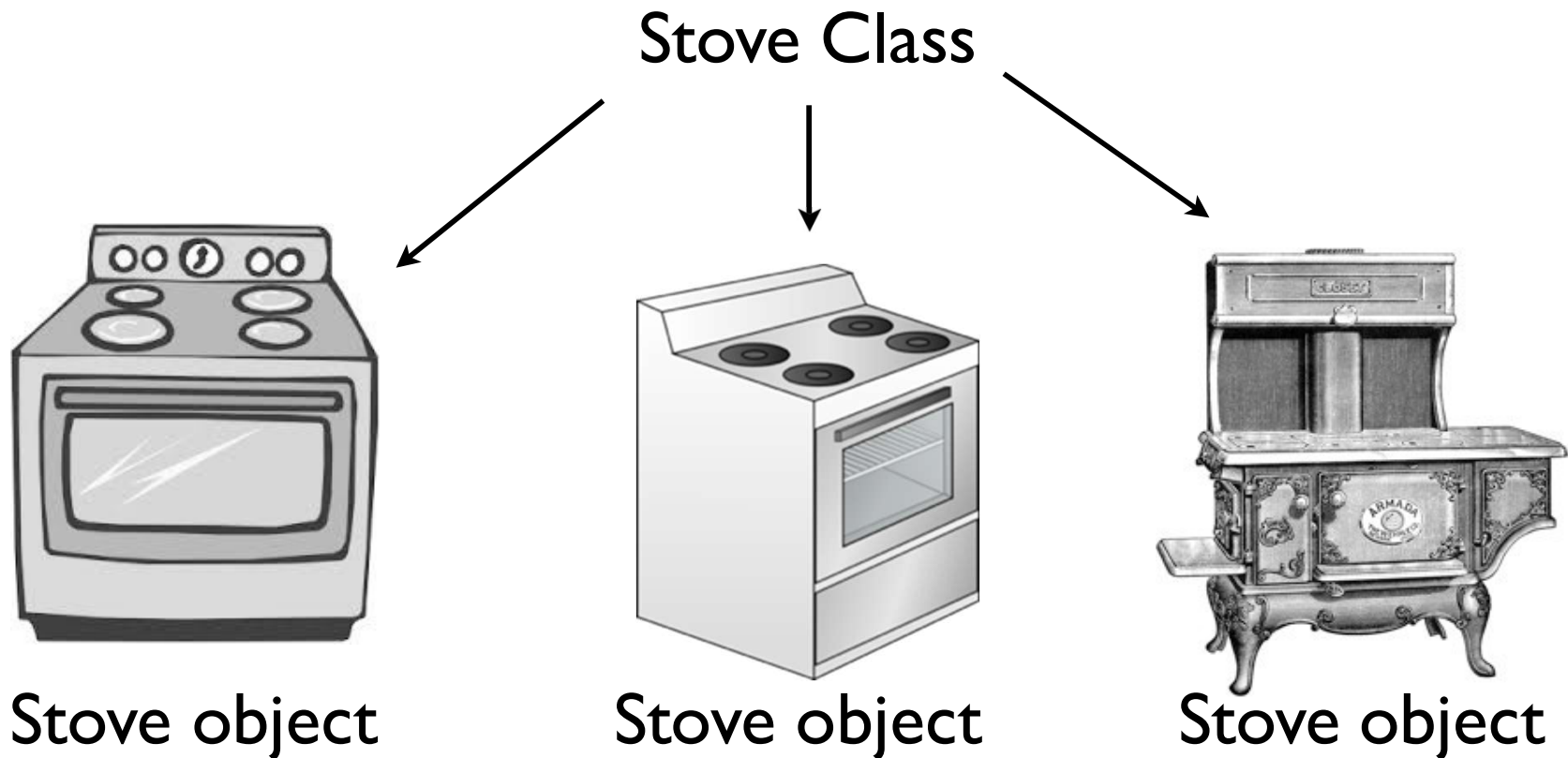
Creating Objects

In Java, we first need a *class* to make an *object*.
A class serves as a blueprint/template for an object.

Stove Class

Creating Objects

In Java, we first need a *class* to make an *object*.
A class serves as a blueprint/template for an object.



- The same class can be used to make different stoves
- These stoves can be different from each other, perhaps even radically different. It all depends on exactly how the class is defined.

`public class`

Declares a `class`, and gives it

`public` visibility (more on that later in the course)

public class

Declares a class, and gives it

`public` visibility (more on that later in the course)

```
public class Table {  
    ...  
}
```

Constructors

Constructors

- Code executed upon object creation
- Effectively create the object
- Looks like a method, but no return type (not even `void`) and has the same name as the class

Constructors

- Code executed upon object creation
- Effectively create the object
- Looks like a method, but no return type (not even `void`) and has the same name as the class

```
public class Table {  
    public Table() {  
        System.out.println(  
            "Creating table...");  
    }  
}
```

Constructors

- Code executed upon object creation
- Effectively create the object
- Looks like a method, but no return type (not even `void`) and has the same name as the class

Constructor

```
public class Table {  
    public Table() {  
        System.out.println(  
            "Creating table...");  
    }  
}
```

Executing Constructors

`new` executes a given constructor,
creating a new object in the process.

Executing Constructors

`new` executes a given constructor,
creating a new object in the process.

```
Table t = new Table();
```

Example:
`Table.java`

Constructor Parameters

Just like methods, constructors can take parameters

Constructor Parameters

Just like methods, constructors can take parameters

```
public class ConsParam {  
    public ConsParam(String str) {  
        System.out.println(str);  
    }  
}
```

Constructor Parameters

Just like methods, constructors can take parameters

```
public class ConsParam {  
    public ConsParam(String str) {  
        System.out.println(str);  
    }  
}
```

```
ConsParam p = new ConsParam("hi");
```

Example:

ConsParam.java

Instance Variables

Instance Variables

Declared in the class.

Each object created from a class (hereafter referred to as an *instance*) has its own instance variables.

Instance Variables

Declared in the class.

Each object created from a class (hereafter referred to as an *instance*) has its own instance variables.

```
public class HasInstance {  
    int myInt; // instance variable  
    ...  
}
```

Instance Variables

Declared in the class.

Each object created from a class (hereafter referred to as an *instance*) has its own instance variables.

```
public class HasInstance {  
    int myInt; // instance variable  
    public HasInstance(int setInt) {  
        myInt = setInt;  
    }  
}
```



```
public class HasInstance {
    int myInt; // instance variable
    public HasInstance(int setInt) {
        myInt = setInt;
    }
}
```

```
public class HasInstance {  
    int myInt; // instance variable  
    public HasInstance(int setInt) {  
        myInt = setInt;  
    }  
}
```

```
HasInstance a = new HasInstance(7);
```

```
public class HasInstance {  
    int myInt; // instance variable  
    public HasInstance(int setInt) {  
        myInt = setInt;  
    }  
}
```

```
HasInstance a = new HasInstance(7);  
HasInstance b = new HasInstance(8);
```

```
public class HasInstance {  
    int myInt; // instance variable  
    public HasInstance(int setInt) {  
        myInt = setInt;  
    }  
}
```

```
HasInstance a = new HasInstance(7);  
HasInstance b = new HasInstance(8);
```

HasInstance a:



```
myInt: 7
```

```
public class HasInstance {  
    int myInt; // instance variable  
    public HasInstance(int setInt) {  
        myInt = setInt;  
    }  
}
```

```
HasInstance a = new HasInstance(7);  
HasInstance b = new HasInstance(8);
```

HasInstance a:

myInt: 7

HasInstance b:

myInt: 8

Example:

HasInstance.java

Instance Methods

Instance Methods

- Define which interactions can occur between objects
- Declared in the `class`
- Specific to objects created from the class (instances), and operate over instance variables.


```
public class HasInstance {
    int myInt; // instance variable
    public HasInstance(int setInt) {
        myInt = setInt;
    }
}
```

-To show an example, let's take the HasInstance definition from before...

```
public class HasInstance2 {
    int myInt; // instance variable
    public HasInstance2(int setInt) {
        myInt = setInt;
    }

    public void printInt() {
        System.out.println(myInt);
    }
}
```

- ...and now we add the printInt instance method
- The name of the class has also been changed, just so we can have both examples in two separate files (namely HasInstance.java and HasInstance2.java)

Example:

HasInstance2.java

static

Associates something with **the class itself**,
as opposed to individual objects created from the class.

static

Associates something with **the class itself**,
as opposed to individual objects created from the class.

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

- You've been defining main and all your methods this way the entire time
- Java forces all source code to be in classes, so this is unavoidable. However, we haven't really gotten into proper objects yet.

static vs. non-static

With static: associated with the class.

Without static: associated with objects
created from the class.

static vs. non-static

With static: associated with the class.

Without static: associated with objects
created from the class.

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

static vs. non-static

With static: associated with the class.

Without static: associated with objects
created from the class.

With class
MyClass

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


static vs. non-static

With static: associated with the class.

Without static: associated with objects
created from the class.

With class
MyClass

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

```
public class MyClassTest {  
    @Test  
    public void someTest() {...}  
}
```

static vs. non-static

With static: associated with the class.

Without static: associated with objects
created from the class.

With class
MyClass

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

With objects created from MyClassTest

```
public class MyClassTest {  
    @Test  
    public void someTest() {...}  
}
```

Stove Example in Java

- `Water.java`
- `Faucet.java`
- `Pot.java`
- `Stove.java`
- `BoilingWater.java`