CENG 1004
Introduction to Object Oriented Programming

WEEK - 4

Today’s Topics

- Lecture 3 Review
- "static" keyword
- Access control
- Class Scope
- Packages
- Java API

Lecture 3 Review

Objects have behaviours

BEHAVIOUR
Barking
Fetching
Eating
Running

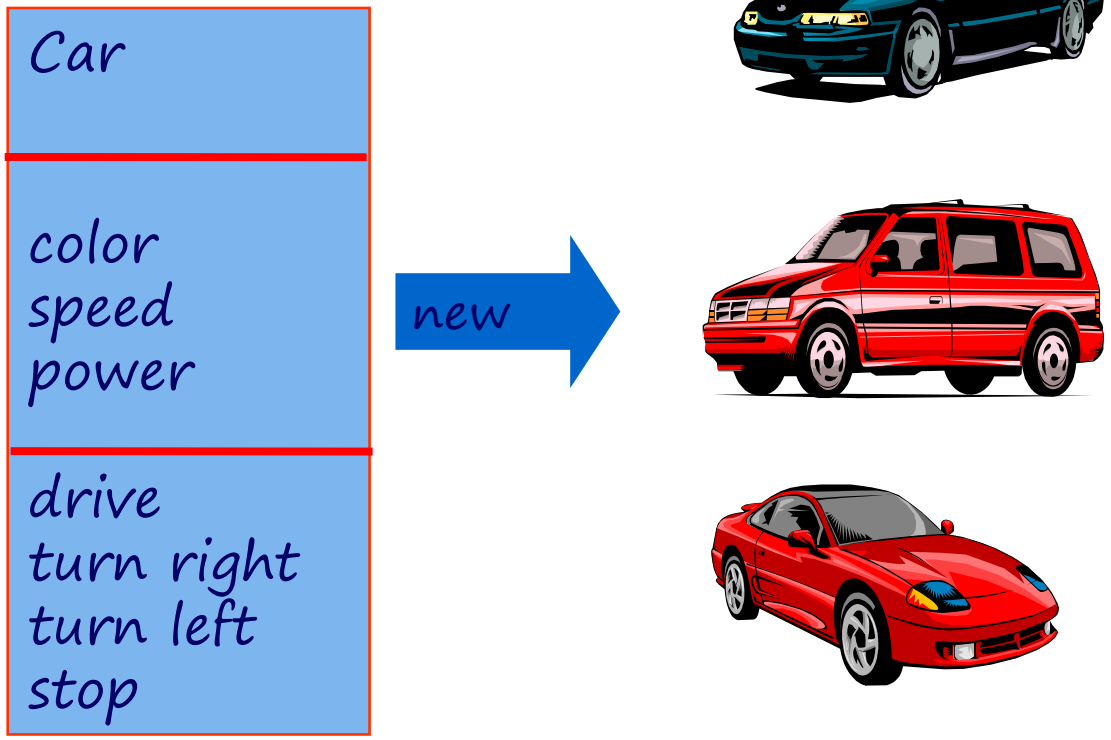
ATTRIBUTES
Name : Pamuk
Color : White
Breed : White Terrier
Hungry: Yes

BEHAVIOUR
Change Gear
Change Direction
Accelerate
Apply Brakes

ATTRIBUTES
Current Gear
Current Direction
Current Speed
Color

Classes

- Serves as template/blueprint from which objects can be created
- Can be used to *create* objects
- Objects are the instances of that class
- Defines attributes and operations



Let's declare a Student

```
public class Student {

}
```

Let's declare a Student

```
public class Student {

    TYPE var_name;
    TYPE var_name = some_value;



}
```

Class - overview

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    String email;

    public void registerCourse(String course){
            courses.add(course);
    }

}
```

Class Definition

Student constructor

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
    }

    public void registerCourse(String course){
        courses.add(course);
    }

}
```

Classes and Instances

```
import java.util.ArrayList;

public class Student {

	String name;
	String id;
	int year;
	ArrayList courses = new ArrayList();
	String email;

	public static void main(String[] args){

		Student student1 = new Student("Ali", 1);
		Student student2 = new Student("Mehmet", 3);
	}

	public Student(String studentName, int studentYear){
		name = studentName;
		year = studentYear;
	}
}
```

Accessing fields

- Object.FIELDNAME

```
Student student1 = new Student("Ali", 1);

System.out.println(student1.name);
System.out.println(student1.year);
```

Calling Methods

- Object.METHODNAME([ARGUMENTS])

Student student1 = new Student("Ali", 1);

student1.incrementYear();
student1.registerCourse("CENG 1004");

References

```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

static

- Keep track of the number of students

```
public class Student {

    String name;
    int year;

    static int count = 0;

    public static double calculateAverageGrade (Student[] students){
            ...
    }
}
```

Questions from last lecture?

static

static field

```
public class Counter {
   int myCount = 0;
   static int ourCount = 0;
   void increment() {
      myCount++;
      ourCount++;
   }
   public static void main(String[] args) {
      Counter counter1 = new Counter();
      Counter counter2 = new Counter();
      counter1.increment();
      counter1.increment();
      counter2.increment();
      System.out.println("Counter 1: " +
counter1.myCount + " " + counter1.ourCount);
      System.out.println("Counter 2: " +
counter2.myCount + " " + counter2.ourCount);
   }
}
```

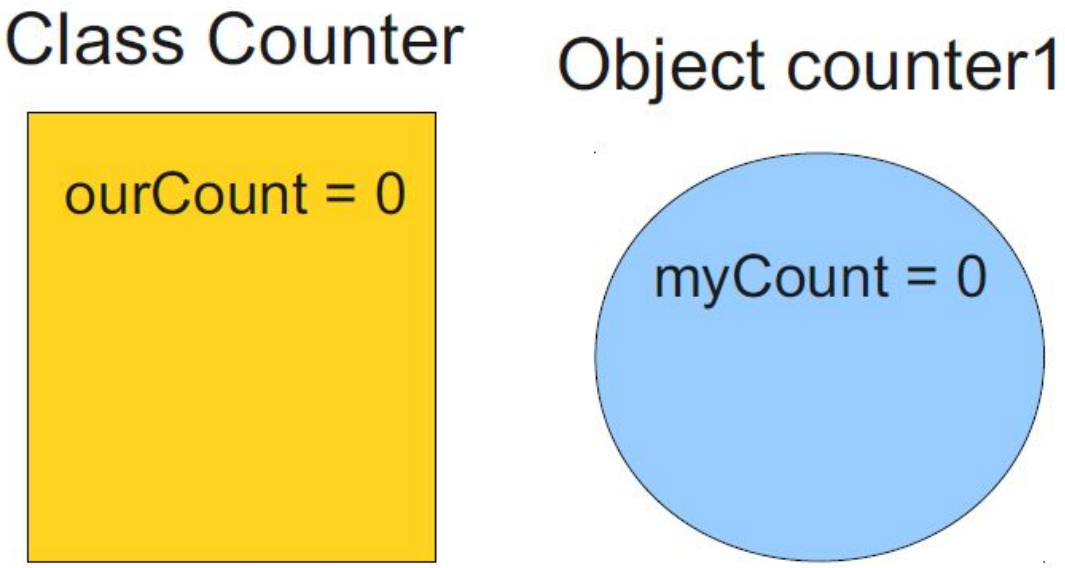
static field

```
public class Counter {
   int myCount = 0;
   static int ourCount = 0;
   void increment() {
      myCount++;
      ourCount++;
   }
   public static void main(String[] args) {
      Counter counter1 = new Counter();
      Counter counter2 = new Counter();
      counter1.increment();
      counter1.increment();
      counter2.increment();
      System.out.println("Counter 1: " +
counter1.myCount + " " + counter1.ourCount);
      System.out.println("Counter 2: " +
counter2.myCount + " " + counter2.ourCount);
   }
}
```

Fields

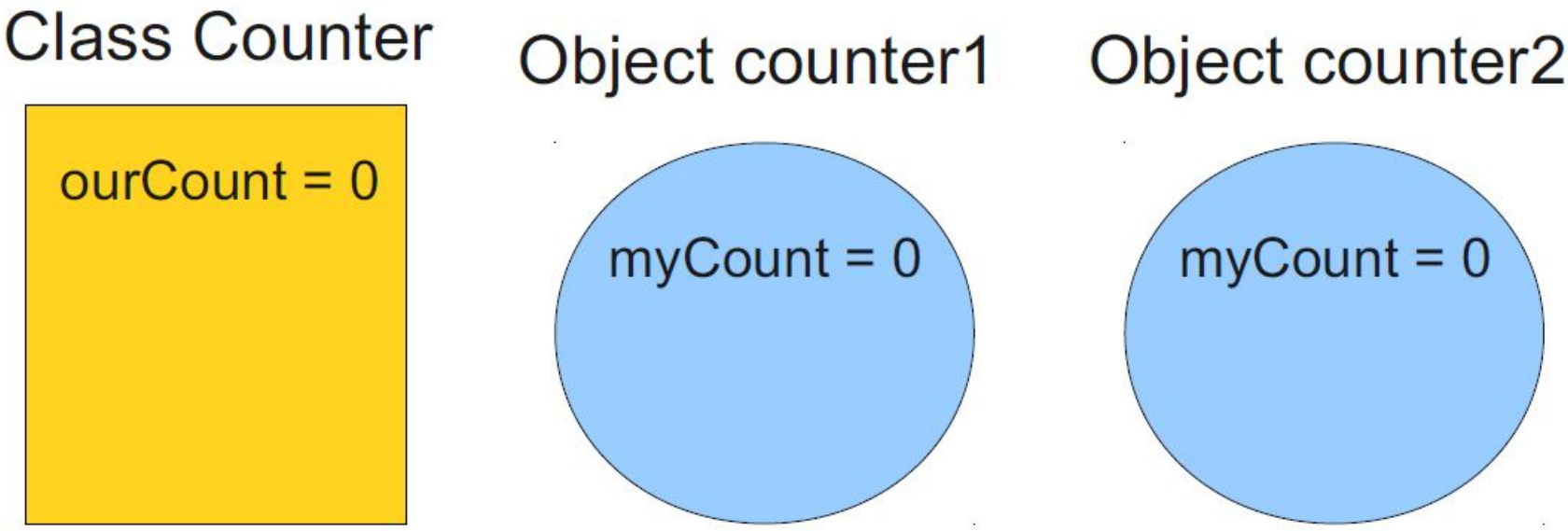
Method

static field



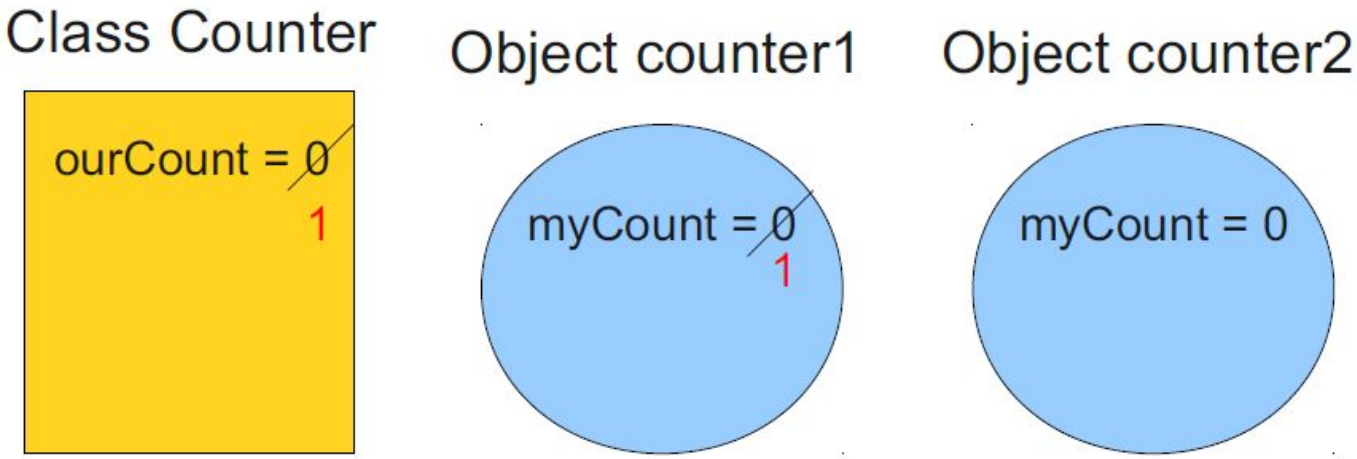

```
Counter counter1 = new Counter();
```

static field



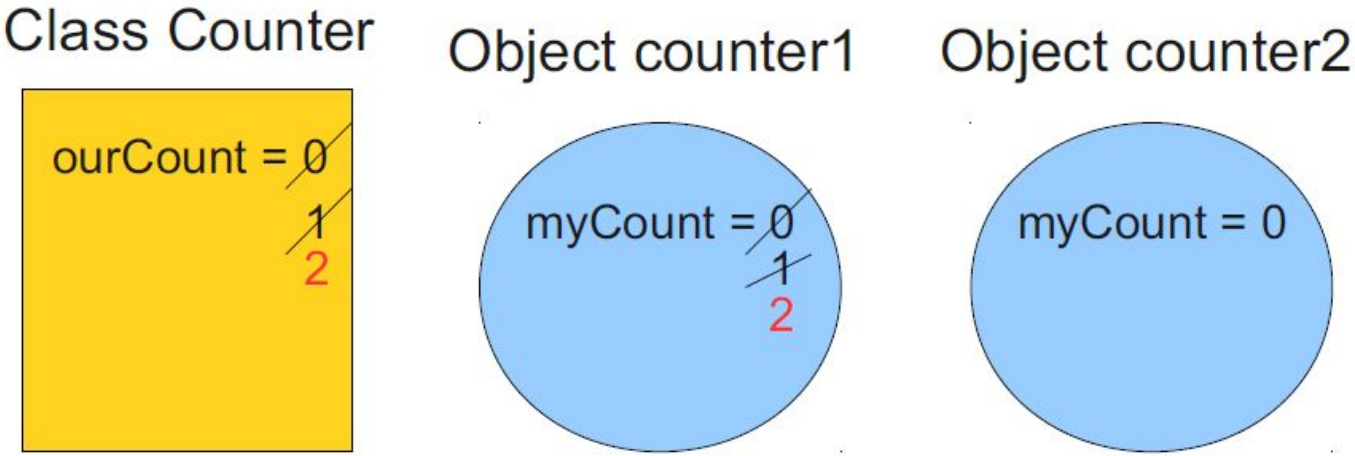

```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
```

static field



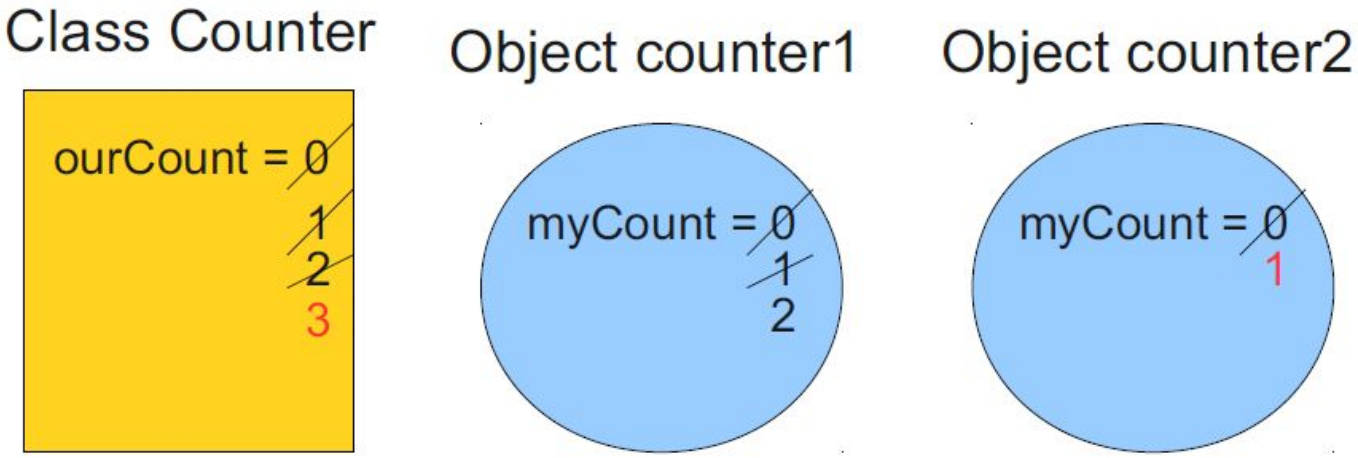

```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
```

static field




```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
counter1.increment();
```

static field




```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
counter1.increment();
counter2.increment();
```

Access control

Access Control

```
public class CreditCard {
   String cardNumber;
   double expenses;
   void charge(double amount) {
      expenses = expenses + amount;
   }
   String getCardNumber(String password) {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
         return "jerkface";
   }
}
```

Malicious

```
public class Malicious {
   public static void main(String[] args) {
      maliciousMethod(new CreditCard());
   }
   static void maliciousMethod(CreditCard card)
   {
      card.expenses = 0;
      System.out.println(card.cardNumber);
   }
}
```

Public vs. Private

- Public: others can use this
- Private: only the class can use this

public/private applies to any
field or **method**

Access Control

```
public class CreditCard {
   String cardNumber;
   double expenses;
   void charge(double amount) {
      expenses = expenses + amount;
   }
   String getCardNumber(String password) {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
         return "jerkface";
   }
}
```

Access Control DONE

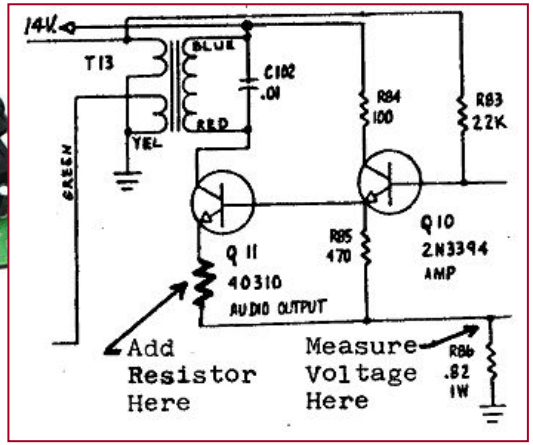
```
public class CreditCard {
   private String cardNumber;
   private double expenses;
   public void charge(double amount) {
      expenses = expenses + amount;
   }
   public String getCardNumber(String password)
   {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
      return "jerkface";
   }
}
```

Why Access Control

- Protect private information
- Clarify how others should use your class
- Keep implementation seperate from interface

Encapsulation

- **encapsulation**: Hiding implementation details of an object from its clients.
  - Encapsulation provides *abstraction*.
    - separates external view (behavior) from internal view (state)
  - Encapsulation protects the integrity of an object's data.



Private fields

- A field can be declared *private*.
  - No code outside the class can access or change it.

```
private type name;
```

  - Examples:

```
private int id;
private String name;
```

- Client code sees an error when accessing private fields:

```
PointMain.java:11: x has private access in Point
System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
                                  ^
```

Accessing private state

- We can provide methods to get and/or set a field's value:

```
// A "read-only" access to the x
field ("accessor")
public int getX() {
    return x;
}

// Allows clients to change the x
field ("mutator")
public void setX(int newX) {
    x = newX;
}
```

Point class

```
// A Point object represents an (x, y) location.
public class Point {
    private int x;
    private int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
}
```

Client code

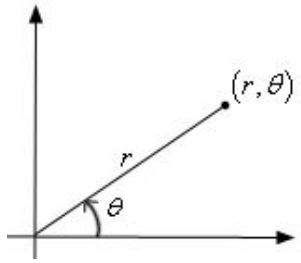
```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1: (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("p2: (" + p2.getX() + ", " + p2.getY() + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2: (" + p2.getX() + ", " + p2.getY() + ")");
  }
}
```

OUTPUT:

```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

Benefits of encapsulation

- Provides abstraction between an object and its clients.
- Protects an object from unwanted access by clients.
  - A bank app forbids a client to change an `Account`'s balance.
- Allows you to change the class implementation.
  - `Point` could be rewritten to use polar coordinates (radius *r*, angle *θ*), but with the same methods.

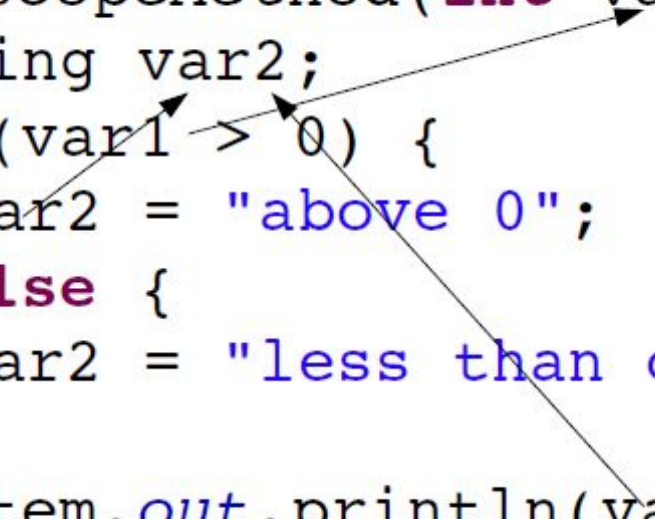


- Allows you to constrain objects' state.
  - Example: Only allow `Point`s with non-negative coordinates.

Class Scope

Scope Review

```
public class ScopeReview {
   void scopeMethod(int var1) {
      String var2;
      if (var1 > 0) {
         var2 = "above 0";
      } else {
         var2 = "less than or equal to 0";
      }
      System.out.println(var2);
   }
}
```



Scope Review

```
public class ScopeReview {
   private int var3;
   void scopeMethod(int var1) {
      var3 = var1;
      String var2;
      if (var1 > 0) {
         var2 = "above 0";
      } else {
         var2 = "less than or equal to 0";
      }
      System.out.println(var2);
   }
}
```

Class Scope

```
public class ScopeReview {
   private int var3;
   void scopeMethod(int var1) {
      var3 = var1;
      String var2;
      if (var1 > 0) {
         var2 = "above 0";
      } else {
         var2 = "less than or equal to 0";
      }
      System.out.println(var2);
   }
}
```

Variable names and scope

- Usually it is illegal to have two variables in the same scope with the same name.

```
public class Point {
    int x;
    int y;
    ...

    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }
}
```

  - The parameters to `setLocation` are named `newX` and `newY` to be distinct from the object's fields `x` and `y`.

Variable shadowing

- An instance method parameter can have the same name as one of the object's fields:

```
// this is legal
public void setLocation(int x, int y) {
    ...
}
```

  - Fields `x` and `y` are *shadowed* by parameters with same names.
  - Any `setLocation` code that refers to `x` or `y` will use the parameter, not the field.

Avoiding shadowing w/ `this`

```
public class Point {
    private int x;
    private int y;

    ...

    public void setLocation(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

- Inside the `setLocation` method,
  - When `this.x` is seen, the *field* `x` is used.
  - When `x` is seen, the *parameter* `x` is used.

'this' keyword

- Clarifies scope
- Means 'my object'

Usage:

```
class Example {
    int memberVariable;
    void setVariable(int newVal) {
        this.memberVariable += newVal;
    }
}
```

Multiple constructors

- It is legal to have more than one constructor in a class.
  - The constructors must accept different parameters.

```
public class Point {
    private int x;
    private int y;

    public Point() {
        x = 0;
        y = 0;
    }

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    ...
}
```

Constructors and `this`

- One constructor can call another using `this`:

```
public class Point {
    private int x;
    private int y;

    public Point() {
        this(0, 0);  // calls the (x, y) constructor
    }

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    ...
}
```

Packages

Packages

- Each class belongs to package
- Classes in the same package serve a similar purpose
- Packages are just directories
- Classes in other packages need to be imported
- Classes that are not explicitly put into a package are in the “default” package.

Packages

Defining Packages

```
package path.to.package.foo;
class Foo {
    ...
}
```

Using Packages

```
import path.to.package.foo.Foo;
import path.to.package.foo.*;
```

Packages

- If the class is in a package named **test.pkg**,
  - then the first line of the source code will be

    package test.pkg;
  - the source code file must be in a subdirectory named “pkg” inside a directory named “test”

    /test/pgk/ClassName.java
  - Use “javac test/pkg/ClassName.java” to compile
  - Use “java test.pkg.ClassName” to execute

Why Packages?

- Group similar functionality
  - org.boston.libraries.Library
  - org.boston.libraries.Book
- Seperate similar names
  - shopping.List
  - packaging.List

Special Packages

- All classes see classes in the same package (No need to import)

- All classes see classes in java.lang
  - Example: java.lang.String; java.lang.System

Java API

Java API

- Java includes lots of packages/classes
- Reuse classes to avoid extra work
- http://docs.oracle.com/javase/8/docs/api/

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

What if the library expands?

ArrayList

Modifiable list

Internally implemented with arrays

Features

- Get/put items by index
- Add items
- Delete items
- Loop over all items

Array → ArrayList

```
Book[] books =
      new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex += 1;
```



```
ArrayList<Book> books
= new ArrayList<Book>();


books.add(b);
```

```
import java.util.ArrayList;
class ArrayListExample {
   public static void main(String[] arguments) {
      ArrayList<String> strings = new ArrayList<String>();
      strings.add("Evan");
      strings.add("Eugene");
      strings.add("Adam");

      System.out.println(strings.size());
      System.out.println(strings.get(0));
      System.out.println(strings.get(1));

      strings.set(0, "Goodbye");
      strings.remove(1);
      for (int i = 0; i < strings.size(); i++) {
            System.out.println(strings.get(i));
       }
      for (String s : strings) {
         System.out.println(s);
      }
   }
}
```

Summary for today

- "static" keyword
- Access control
- Class Scope
- Packages
- Java API

References

- http://math.hws.edu/javanotes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/
- https://docs.oracle.com/javase/tutorial/java
- https://courses.cs.washington.edu/courses/cse142/11au/lectures/11-23/23-ch08-3-encapsulation.ppt