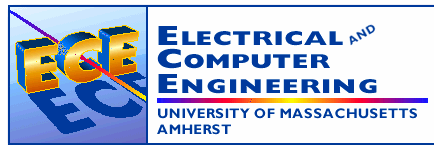

ECE 122

Engineering Problem Solving with Java

Lecture 15

Class Relationships



Outline

- **Problem: How can I create and store complex objects?**

- **Review of static methods**
 - Consider static variables

- **What about objects that are stored in other objects**
 - How can I access them?

- **Moving towards complicated objects**

The static Modifier

- → We declare static methods and variables using the static modifier
- It associates the method or variable with the class rather than with an object of that class
- → Static methods are sometimes called *class methods*
 - Static variables are sometimes called *class variables*
- What happens in memory for static variables and methods?

Static Variables

- Normally, each object has its own data space
 - If a variable is declared as static, only one copy of the variable exists
- ```
private static float price;
```
- Memory space for a static variable is created when the class is first referenced!
  - All objects instantiated from the class share its static variables
  - → Changing the value of a static variable in one object changes it for all others
    - Need to use static variables very carefully

## Static variables examples

---

- **Static** variables are shared across all instances of a class.
- They are usually associated with the class itself, rather than a “shared space”

```
public class Particle {
 public static final int X = 0, Y = 1, Z = 2;
 ..
 public int getX () {
 return position[X]; // use as index
 }
 ..
}
```

## Static Class Members

---

- A static method is one that can be invoked through its class name
- For example, the methods of the `Math` class are static:

```
result = Math.sqrt(25)
```

- ➔ Variables can be static as well
- Determining if a method or variable should be static is an important design decision
  - What does *static* mean to you?

## Static methods

---

- Can only access static fields
- No object variable is passed as an implicit parameter
- Can be good for utility methods

```
Math.abs (247);
Integer.parseInt ("-37");
```

## Static Methods

---

```
class Helper
{
 → public static int cube (int num)
 static method {
 (class method) return num * num * num;
 }
} Because it is declared as static, the method
can be invoked as

value = Helper.cube (5);
```

↑  
Note: Helper is a class. The cube method is invoked via class name.class method, since NO object needs to be instantiated to use a static method..

Note: the method returns an int...

## Static Class Members

---

- **The order of the modifiers can be interchanged, but by convention visibility modifiers come first**
  - E.g. public, private
- **Recall that the `main` method is static**
- **Static methods cannot reference instance variables**
  - Instance variables don't exist until an object exists
  - Instance data is unique to each object!
- **Static method can reference static variables or local variables**

## Static Class Members

---

- **Static methods and static variables often work together**
- **→ The following example keeps track of how many `Slogan` objects have been created using a static variable, and makes that information available using a static method**
- **→ This is a very popular use of static variables:**
  - *Counting the number of objects of a particular class and providing a static method to get (print?) that count out when needed.*

## Class Relationships – Essential Concept!

- **Classes in a software system can have various types of relationships to each other**
- **Three of the most common relationships:**
  - **Dependency: A *uses* B**
  - **Aggregation: A *has-a* B**
  - **Inheritance: A *is-a* B**
- **Let's discuss dependency and aggregation further**
- **Inheritance is discussed in detail in Chapter 8**

## Dependency

- **→ A *dependency* exists when one class relies on another in some way,**
  - **Usually involves invoking the methods of the other (e.g. `System.out.println()`...)**
- **We've seen dependencies in many previous examples**
- **→ We don't want numerous or complex dependencies among classes**
- **Alternately, some dependence is OK since we need to build a**
- **A good design strikes the right balance**

## Dependency

---

- → Some dependencies occur between objects of the same class
- A method of the class may accept an object of the same class as a parameter
- For example, the `concat` method of the `String` class takes as a parameter another `String` object

```
str3 = str1.concat(str2);
```

- This drives home the idea that the service is being requested from a particular object
- Recall the format of the `String` methods. They are almost all quite similar: `object.method(object)`...

## Dependency

---

- The following example defines a class called `Rational` to represent a rational number
- A rational number is a value that can be represented as the ratio of two integers
- Some methods of the `Rational` class accept another `Rational` object as a parameter

## Aggregation

---

- An *aggregate* is an object that is made up of other objects
- Therefore aggregation is a *has-a* relationship
  - A car *has a* chassis
- In software, an aggregate object contains references to other objects as instance data
- The aggregate object is defined in part by the objects that make it up
- Basically the object contains instances of other objects

## The this Reference

---

- The `this` reference allows an object to refer to itself
- That is, the `this` reference, used inside a method, refers to the object through which the method is being executed
- Suppose the `this` reference is used in a method called `tryMe`, which is invoked as follows:

```
obj1.tryMe();
obj2.tryMe();
```

- In the first invocation, the `this` reference refers to `obj1`; in the second it refers to `obj2`

## The this reference

---

- The `this` reference can be used to distinguish the instance variables of a class from corresponding method parameters with the same names
- The constructor of the `Account` class (from Chapter 4) could have been written as follows:

```
public Account (String name, long acctNumber,
 double balance)
{
 this.name = name;
 this.acctNumber = acctNumber;
 this.balance = balance;
}
```

## Summary

---

- **Moving toward more complicated objects**
  - Using the same object type in an object
- **Static variables and methods have important functions**
- **Important to consider the hierarchical use of objects**
- **Building systems increases the focus on problem solving**