Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Abstract classes and methods
- Final
- Static

Today's Lecture

- You can make classes whose sole purpose is to be inherited from.
- These are called "**abstract**" classes.
- An abstract class can hold code that has common behavior or data.
- You **cannot** make an instance of an abstract class.

abstract

Coding with Inheritance

- Write programs that process objects that share the same base class in a class hierarchy.
- The base class contains the common behavior that we care about.
- ***Put common behavior in base class.***
- ***Program to the common behavior.***

Why Create an Abstract Class

- What if it does not make sense to create instances of the base class.
- For example, think about Shape, Rectangle, and Circle.
- It does not make sense to create an instance of Shape because it is too general or abstract.
- In this case we can make Shape an **abstract** base class.
- An abstract class still defines the common behavior that we care about.
- We just cannot create instances of an abstract class.

- An abstract class is like a "template".
- In MS Word you have different templates for different types of documents.
- In MS Word you use the template as a starting off point for your document. The template itself is not a finished product.

Abstract Class

- Now revisit the employee salary example.
- We will now make the Employee class abstract.
- For example…

Abstract Class Example

```
public abstract class Employee
{
   protected double salary;

   public Employee(double newSalary)
   { salary = newSalary; }

   public double GetSalary()
   { return salary; }

   public void SetSalary(double newSalary)
   { salary = newSalary; }

   public abstract void ShowWeeklySalary(); // Derived classes
}                                           //MUST override this
```

Abstract Class Example

Abstract Method

- An abstract class can have abstract methods (as well as normal methods).
- There is one abstract method in Employee:

public abstract void ShowWeeklySalary();

- An abstract method does NOT have a body.
- It only lists the method name, parameters, and return type.

- Derived classes must give a definition for all abstract methods on its base class (basically).
- The only way for a derived class to not give a definition would be to also make the derived class abstract.

```
public class HourlyEmployee extends Employee
{
   public HourlyEmployee(double newSalary)
   {
         super(newSalary);
   }

   // OVERRIDE Employee::ShowWeeklySalary()
   @Override
   public void ShowWeeklySalary()
   {
         double weeklySalary = salary * 40;

         System.out.printf("Hourly Rate   = $%.2f\n", salary);
         System.out.printf("Weekly Salary = $%.2f\n",weeklySalary);
   }
}
```

HourlyEmployee MUST give a definition for ShowWeeklySalary or it will not compile

Derive from an Abstract Class

```
public class SalaryEmployee extends Employee
{
    public SalaryEmployee(double newSalary)
    {
            super(newSalary);
    }

    // OVERRIDE Employee::ShowWeeklySalary()
    @Override
    public void ShowWeeklySalary()
    {
            double weeklySalary = salary / 52.0;

            System.out.printf("Yearly Rate   = $%.2f\n", salary);
            System.out.printf("Weekly Salary = $%.2f\n",weeklySalary);
    }
}
```

Derive from an Abstract Class

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

  //Employee e = new Employee(30); // NOT ALLOWED.
                                   // Employee is abstract!!!

  Employee e1;
  e1 = new SalaryEmployee(52000);

  Employee e2;
  e2 = new HourlyEmployee(20);

  e1.ShowWeeklySalary();
  e2.ShowWeeklySalary();
}
```

You can declare a variable of an abstract class type

Create instance of derived class and put the reference in an abstract class variable

You can call an abstract method. It will use the ShowWeeklySalary definition for the underlying type:
e1 → Calls SalaryEmployee version
e2 → Calls HourlyEmployee version

Use Abstract Class

- Now on to final…

final

final Class

- Can you prevent a class from being inherited from?
- Yes.
- You must declare the class as "final".
- A "final" class cannot be inherited from.

```
public final class Employee {
        // Employee members go here…

}
```

~~public class Manager extends Employee {~~
~~// Manager members go here…~~

~~}~~

- Cannot inherit from the Employee class since its final.

final Class

final Method

- Can you prevent a method from being overridden?
- Yes.
- You must declare the method as "final".
- A final method cannot be overridden.

```
public class Employee {
   public final void myMethod() {
      System.out.println("myMethod called");
   }
}

public class Manager extends Employee {
   @Override
   public void myMethod() {
   }
}
```

- Cannot override a method that is defined as final.

final Method

- Now on to static…

Static

Static

- Both methods and variables can be declared ***static***.
- A static variable is shared by every instance of the class.
  - There is only 1 copy of a static variable in memory.
- If you make a change from one instance you will "see" that change in another instance.
- Use a static variable if you *don't* need a different version of that variable for EVERY instance of the class.

- INSTANCE VARIABLES:
- Suppose a student class:

```
public class Student {
   public int id;
   public int rank;

   public Student(int newId, int newRank) {
         id = newId;
         rank = newRank;
   }
}
```

Static

- INSTANCE VARIABLES:
- 3 Instances of student. Each has its own set of variables.

```
Student s1, s2, s3;

s1 = new Student(12, 3);
s2 = new Student(10, 100);
s3 = new Student(7, 70);
```

s1:Student	s2:Student	s3:Student
12(id)	10 (id)	7 (id)
3 (rank)	100 (rank)	70 (rank)

Static

- Now suppose we want to store a count of the total number of students.
- The number of students is not specific to any instance so it should be defined as static.
- For example…

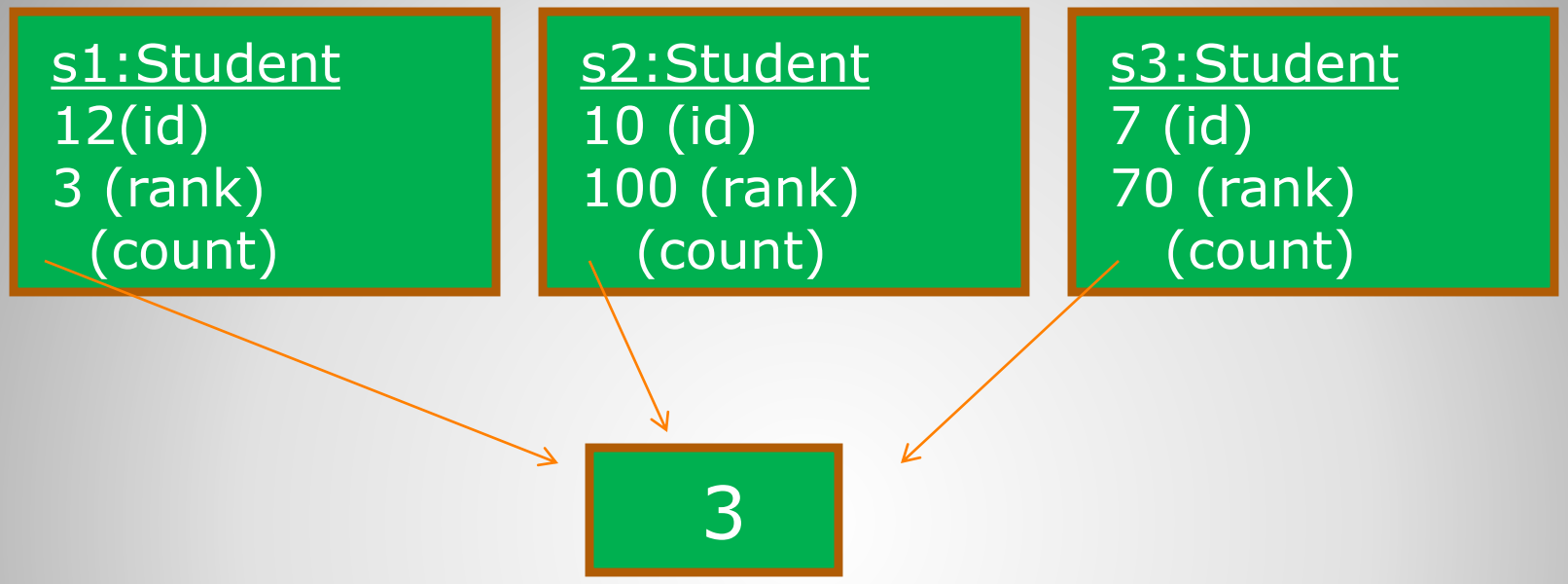
Static

```
public class Student {
   int id;
   int rank;
   static int count;

   public Student(int newId, int newRank) {
         id = newId;
         rank = newRank;
   }
}
```

Use the static keyword to declare a variable as static.

Static



- A static variable is shared by all instances.

Static

- You can access a static variable even if you do not declare an instance of the class.
- Use the class name followed by a dot and then the variable name.

For example:

Student.count = 3;

This sets the count variable to 3.

Static

- Now on to static methods…

Static Methods

- Methods can also be declared as static.
- Static methods can only access static variables within a class.
- Static methods CANNOT access instance variables or instance methods within a class.

Static Method Access Within a Class

- Why is main() declared static?
- main is static so that the JVM can call it without creating an object.
- The JVM will automatically call the static main method when the program starts.
- Regular instance methods can only be used from an instance of a class so main needs to be static if the JVM needs to call it without creating an object.

Static Method - main

Accessing Instance Variables

- Only instance methods can access instance variables within a class.
- In the code below num is an instance variable.
- num cannot be accessed from main because main is static.
- otherNum is static and can be accessed from main.

```
public class Main {
    public int num;                 // Instance variable
    public static otherNum;         // static variable

    public static void main(String[] args) {
        num = 1;

        otherNum = 1;
    }
}
```

Main is a static method so it cannot access an instance variable

otherNum can be accessed from main because it is a static variable.

Calling Static vs Instance Methods

- Cannot call an instance method from a static method within a class

```
public class Main {
    public void myInstanceMethod() {
        System.out.println("myInstanceMethod called");
    }

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

Main is a static method. You cannot call an instance method from a static method.

- Instance methods must be called with respect to an instance of a class (shown on an upcoming slide).

- You can call a static method from another static method within a class

```
public class Main {
    public static void myStaticMethod() {
        System.out.println("myStaticMethod called");
    }

    public static void main(String[] args) {

        myStaticMethod();

    }
}
```

This method call works because myStaticMethod is defined as static

Call Static Method Within a Class

- Instance methods must be called with respect to an instance when inside of a static method.

Instance method (not static)

```
public class Main {
    public void myMethod() {
        System.out.println("myMethod called");
    }

    public static void main(String[] args) {

        Main m = new Main();
        m.myMethod();
    }
}
```

Instance methods must be called with respect to an instance (m is the instance)

- Instance methods must be called with respect to an instance of the class (next slide for this).

Call Instance Method

```
public class X {
    public int num;

    public void method1() {
        // method1 code goes here…
    }

    public void method2() {
        method1();
    }
}
```

Can we do this?

We are trying to call an instance method, but it is not being done with respect to an object.

Call Instance Method

```
public class X {
    public int num;

    public void method1() {
        // method1 code goes here…
    }

    public void method2() {
        method1();
    }
}
```

YES!!! You can do this.

method1 is being called with respect to the this reference. Since method2 is an instance method (not static) it will have a this reference when it is called.

Call Instance Method

- End of Presentation

End of Presentation