# CS 251
# Intermediate Programming
# More on classes

Brooke Chenoweth

University of New Mexico

Spring 2025

# Empty Class

```
public class EmptyClass {
}
```

- Has inherited methods and fields from parent (in this case, `Object`)
- Has default constructor

# Empty Interface

```
public interface EmptyInterface {
}
```

- Often used to "tag" a class.
- Java API has several of these *marker interfaces*
  - `java.lang.Cloneable`
  - `java.io.Serializable`
  - `java.util.EventListener`

# Nested classes

```
public class OuterClass {
  private class InnerClass {
  }
  public static class StaticNestedClass {
  }
}
```

- A nested class is a member of its enclosing class.
- Can be `private`, `protected`, `private`, or package private.
- Non-static nested classes are known as *inner classes*
- Inner classes have access to other members of enclosing class, even if private.

# Why use Nested Classes?

- Logically group classes used in only one place. Nesting a "helper class" keeps it together with the class that uses it.
- Encapsulation. Nested class can access private members of outer class. Nested class also can be private.
- More readable and maintainable code. Nesting small classes within top-level classes places the code closer to where it is used.

# Static Nested Classes

- Behaves like a top level class that is nested in another top level class for packaging convenience.
- Like a static method, cannot directly access instance variables or methods in enclosing class. Need object reference.
- Access using enclosing class name when using in another class

```
OuterClass.StaticNestedClass nestedObject =
    new OuterClass.StaticNestedClass();
```

# Inner Classes

- Associated with instance of enclosing class and has access to that object's methods and fields.
- Cannot define any static members itself.
- Must have instance of outer class to instantiate an inner class.

```
OuterClass.InnerClass innerObject =
    outerObject.new InnerClass();
```

(Usually just use the implicit `this` object when calling from non-static method)

- Special kind of inner classes: *local classes* and *anonymous classes*

# Shadowing

```
public class ShadowDemo {
  public int x = 0;
  public class Inner {
    public int x = 1;

    public void methodInInner(int x) {
      System.out.println("x = " + x);
      System.out.println("this.x = " + this.x);
      System.out.println("ShadowDemo.this.x = "
        + ShadowDemo.this.x);
    }
  }
  public static void main(String[] args) {
    ShadowDemo sd = new ShadowDemo();
    ShadowDemo.Inner in = sd.new Inner();
    in.methodInInner(23);
  }
}
```

Output:

```
x = 23
this.x = 1
ShadowDemo.this.x = 0
```

# Local Classes

A *local class* is an inner class defined inside a method body.

```
public class LocalClassExample {

  public static void sayHello() {

    class HelloHelper() {
      public void printHello() {
        System.out.println("Hello!");
      }
    }

    HelloHelper myHelper = new HelloHelper();
    myHelper.printHello();
  }
}
```

# Anonymous Classes

An *anonymous class* is a local class defined without a name.

```
public class AnonymousClassExample {
  public interface Greeter {
    void greet();
  }
  public void sayHello() {
    Greeter greeting = new Greeter() {
      public void greet() {
        System.out.println("Hello!");
      }
    };
    greeting.greet();
  }
```

Anonymous classes are often used in GUI applications.

# Simple Enum example

```
public enum Day {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
  THURSDAY, FRIDAY, SATURDAY
}
```

```
public static void main(String[] args) {
  Day today = Day.FRIDAY;
  switch(today) {
    case SATURDAY:
    case SUNDAY:
      System.out.println("Yay! Weekend!");
      break;
    default:
      System.out.println("Not another weekday!");
      break;
  }
}
```

# Enums are classes

- All enums implicitly extend `java.lang.Enum`
- Can define fields and methods
  - More powerful than enums in other languages.
  - Allows data and behaviour to be packaged with the enum constants.
- Constructor must be private or package-private
- Cannot explicitly call constructor

# Fancier Enum Example

```
public enum Day {
  SUNDAY(true), MONDAY(false), TUESDAY(false),
  WEDNESDAY(false), THURSDAY(false),
  FRIDAY(false), SATURDAY(true);

  private final boolean isWeekend;
  private Day(boolean isWeekend) {
    this.isWeekend = isWeekend;
  }

  public void exclaim() {
    if(isWeekend) {
      System.out.println("Yay! Weekend!");
    } else {
      System.out.println("Not another weekday!");
    }
  }
}
```

# Enum Example: Planets

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    private final double mass;   // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() { return mass; }
    private double radius() { return radius; }
```

# Planets continued

```
    // universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
    public static void main(String[] args) {
        double earthWeight = 175;
        double mass =
          earthWeight/EARTH.surfaceGravity();
        for (Planet p : Planet.values())
           System.out.println("Your weight on " + p
               + " is " + p.surfaceWeight(mass));
    }
}
```

# Planets output

```
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
```

The planets example is from the Java Tutorial.