Lab Exercise: Inheritance

Instructors Who Consult. | Consultants Who Teach.

About Intertech

Thank you for choosing Intertech for your training. The next page of this document will get you started with your lab if you'd like to skip ahead. Below is a brief overview of Intertech as well as a promo code for you for future live Intertech trainings.

Intertech (www.intertech.com) is the largest combined software developer **training** and **consulting** firm in Minnesota. Our unique blend of training, consulting, and mentoring has empowered technology teams in small businesses, Fortune 500 corporations and government agencies since 1991.

Our training organization offers live in-classroom and online deliveries, private on-site deliveries, and on-demand options. We cover a broad spectrum of .NET, Java, Agile/Scrum, Web Development, and Mobile technologies. See more information on our training and search for courses by **clicking here**.

We appreciate you choosing Intertech!

Lab Exercise

Inheritance

Inheritance is a powerful mechanism to provide reuse among type definitions. In this lab, you use the "extends" keyword to build classes that inherit from another.

Specifically, in this lab you will:

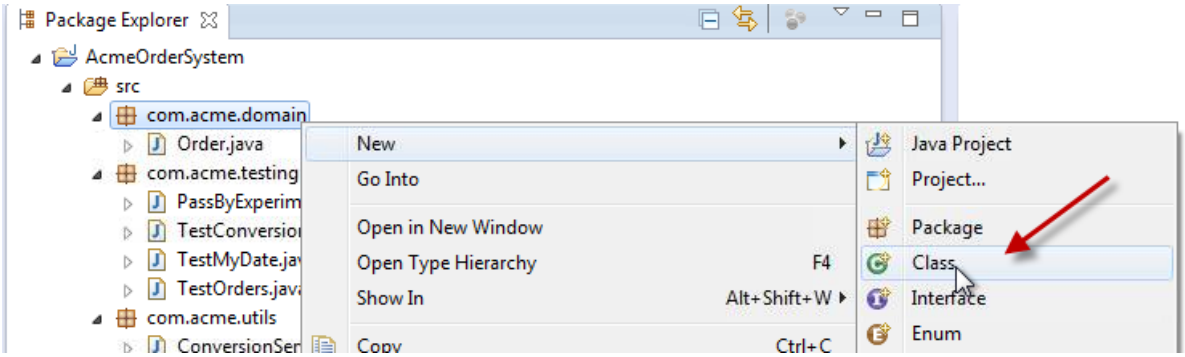
- Create a hierarchy of product types using inheritance
- Use "super" to chain constructors in the class hierarchy
- Explore overriding of super class methods
- See a public static final constant on one of Java's classes
- Optionally look at a "final" method

Scenario

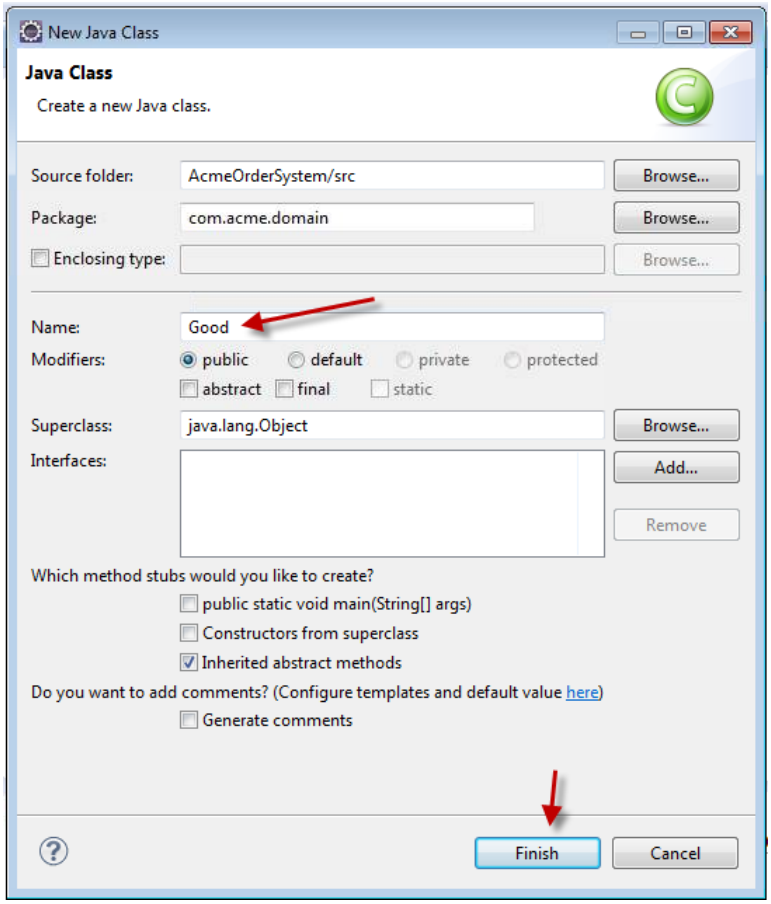
Currently, a simple String is used to indicate the product purchased as part of an order. The user community wants more detail associated with each product, especially as it relates to flammability, hazardousness, product dimensions, and weight, and so on, all characteristics important to the shipment of orders. Some properties are shared across all types of product. For example, all products have a name and model number. Unfortunately, not all products have an identical set of characteristics. Some products are in liquid form, while others are solid. In this lab, you create a set of classes to represent the various products in the Acme line.

Step 1: Create the base product class

1.1 Create the Good class. In the Package Explorer view, right-click on the com.acme.domain package in the AcmeOrderSystem project and select ***New > Class***.



1.2 In the New Java Class window, enter Good as the new class name and click the ***Finish*** button.



1.3 In the Good.java editor view, add the common fields for all products, namely, private field for name, modelNumber, height, unitOfMeasure (an enum), flammable, and weight per unit of measure.

```
public enum UnitOfMeasureType {LITER, GALLON, CUBIC_METER,
CUBIC_FEET}
private String name;
private int modelNumber;
private double height;
private UnitOfMeasureType unitOfMeasure;
private boolean flammable = true;
```

```
private double weightPerUofM;
```

1.4 Properly encapsulate these properties by generating public getter and setter methods for each.

1.5 Add a constructor to create a Good with a name, model number, height, unit of measure, and flammability.

```
public Good(String name, int modelNumber, double height,
UnitOfMeasureType uoM, boolean flammable, double wgtPerUoM) {
  this.name = name;
  this.modelNumber = modelNumber;
  this.height = height;
  this.unitOfMeasure = uoM;
  this.flammable = flammable;
  this.weightPerUofM = wgtPerUoM;
}
```

1.6 Add a toString( ) method to Good. The toString( ) method for Good should display the name and model number for Good.

```
public String toString() {
  return name + "-" + modelNumber;
}
```

1.7 Add a method to provide the volume of the good. Volume varies based on the dimensions of the good. In the case of a generic good, return zero as the volume.

```
public double volume() {
  return 0.0;
}
```

1.8 Add a method to provide the weight of the good. The weight is equal to the volume times the weight per unit of measure.

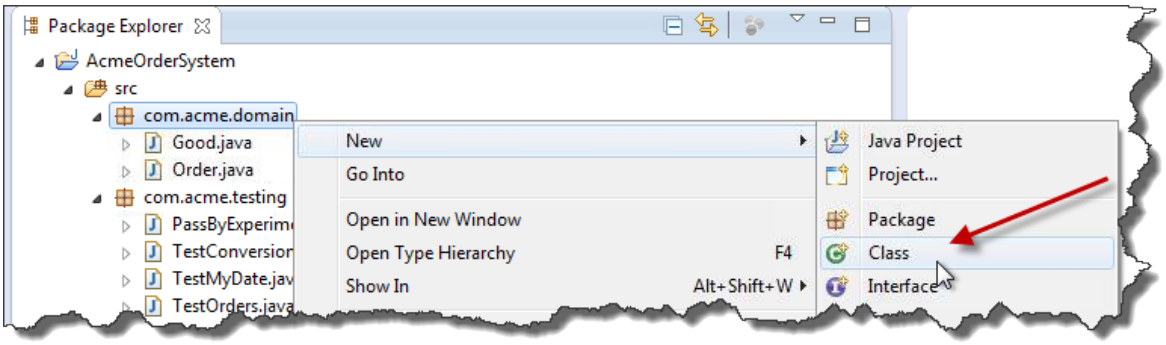
```
public double weight() {
  return volume() * weightPerUofM;
}
```

1.9 Save the Good.java file once you have finished coding, and ensure there are no compiler errors.

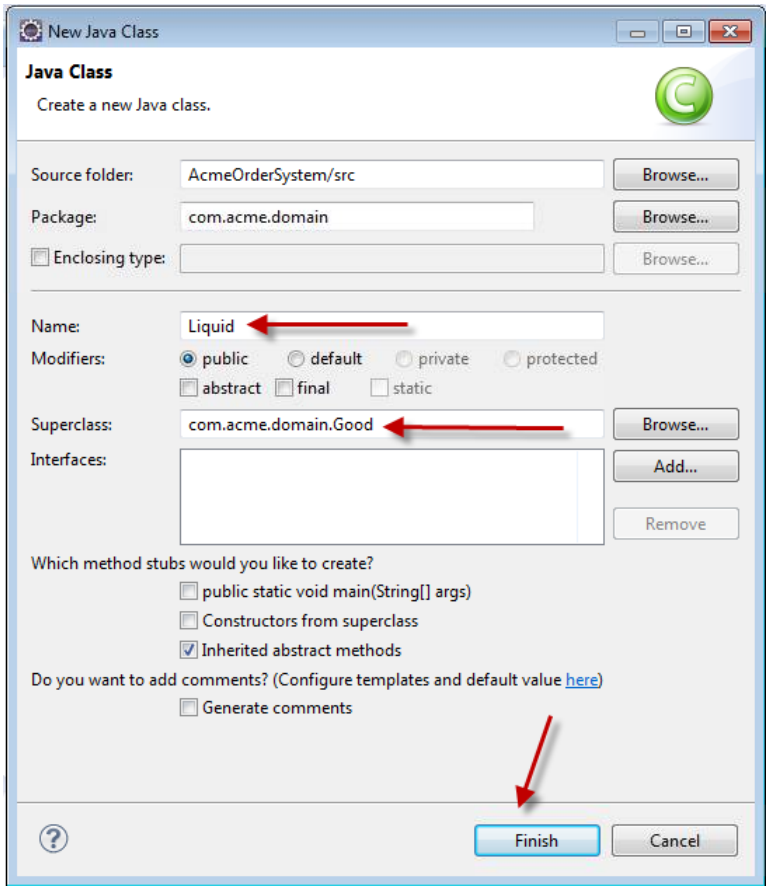
Step 2: Create the Liquid subclass

The Liquid goods class shares the same properties and methods but also provides some additional information such as volume, weight, and boiling point.

2.1 Create the Liquid class. In the Package Explorer view, right-click on the com.acme.domain package in the AcmeOrderSystem project and select ***New > Class***.



2.2 In the New Java Class window, enter Liquid as the new class name, and make sure that Good replaces java.lang.Object as the Superclass name. Click the ***Finish*** button when the information has been provided.



Note: Notice the use of the "extends" keyword in the new Liquid class that is created and displayed in an editor view.

2.3 In the Liquid.java editor view, add the specific fields of liquid products, namely, a private field for the radius of the liquid's container.

```
private double radius;
```

2.4 Properly encapsulate this new property by generating a public getter and setter method.

2.5 Add a Liquid constructor that reuses the super-class constructor (Good), but adds the additional radius parameter.

```
public Liquid(String name, int modelNumber, double height,
UnitOfMeasureType uoM, boolean flammable, double wgtPerUofM,
double radius) {
  super(name, modelNumber, height, uoM, flammable, wgtPerUofM);
  this.radius = radius;
}
```

2.6 Override the Good class's volume( ) method to provide the volume for a cylinder of the liquid.

```
public double volume() {
  return Math.PI * radius * radius * getHeight();
}
```

Note: Try using just the height field in the volume( ) method above. It won't work and gives you a compile error. Do you know why?

Note: Did you notice the use of Math.PI in the calculation of volume? What is Math.PI? This is a public static ***final*** variable defined on the Math class.

2.7 Override the super class's toString( ) method. The toString( ) method for a Liquid should display the name and model number, as well as the fact that it is a liquid and its volume.

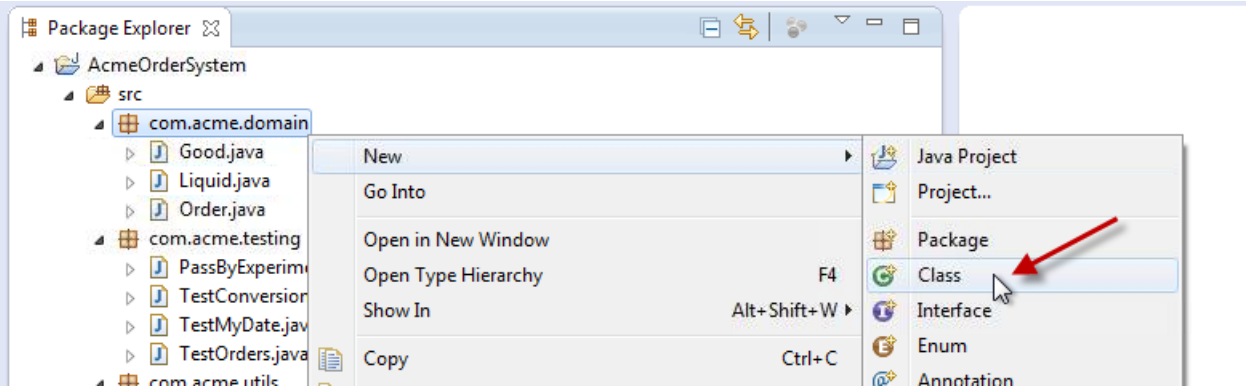
```
public String toString() {
  return super.toString() + " (liquid) " + volume() + " " +
getUnitOfMeasure();
}
```

2.8 Save the Liquid.java file once you have finished coding, and ensure there are no compiler errors.

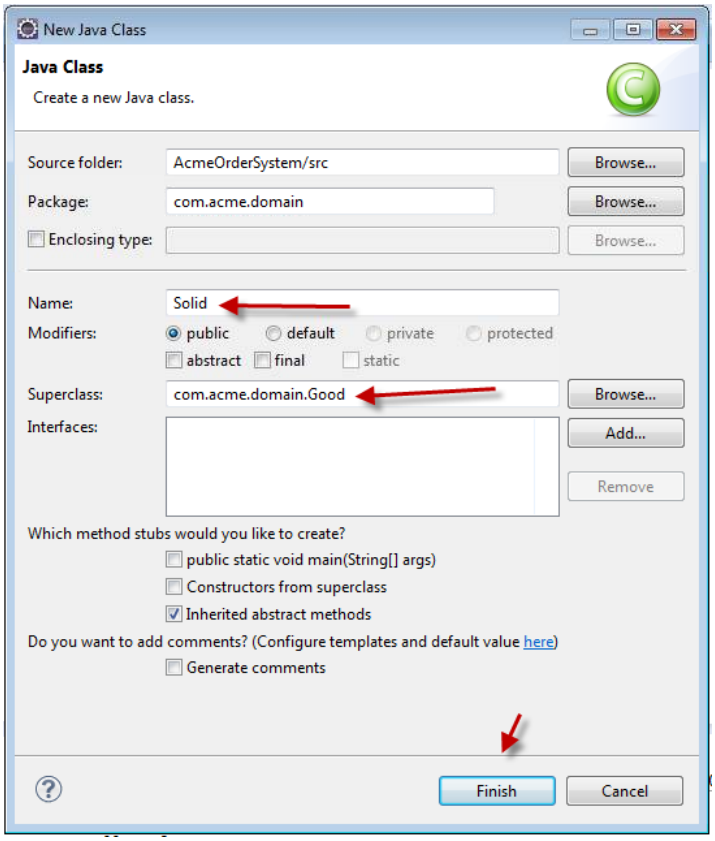
Step 3: Create the Solid subclass

The solid goods class shares the same properties and methods with the Good class but also provides some additional information such as width, length, and melting point.

3.1 Create the Solid class. In the Package Explorer view, right-click on the com.acme.domain package in the AcmeOrderSystem project and select ***New > Class***.



3.2 In the New Java Class window, enter Solid as the new class name, and again make sure that Good replaces java.lang.Object as the Superclass name. Click the ***Finish*** button when the information has been provided.



3.3 In the Solid.java editor view, add the specific fields of solid products, namely, private fields for width, length, and the solid's melting point.

```
private double width;
private double length;
```

3.4 Properly encapsulate these new properties by generating public getter and setter methods for each.

3.5 Add a Solid constructor that reuses the super-class constructor (Good) but adds the additional width and length parameters.

```
public Solid(String name, int modelNumber, double height,
UnitOfMeasureType uoM, boolean flammable, double wgtPerUofM,
double width, double length) {
  super(name, modelNumber, height, uoM, flammable, wgtPerUofM);
  this.width = width;
  this.length = length;
}
```

3.6 Override the Good class's volume( ) method to provide the volume( ) for the container of the solid.

```
public double volume() {
  return width * length * getHeight();
}
```

3.7 Override the super class's toString( ) method. The toString( ) method for a Solid should display the name and model number, as well as the volume of the product.

```
public String toString() {
  return super.toString() + " that is " + volume() + " " +
getUnitOfMeasure() + " in size";
}
```

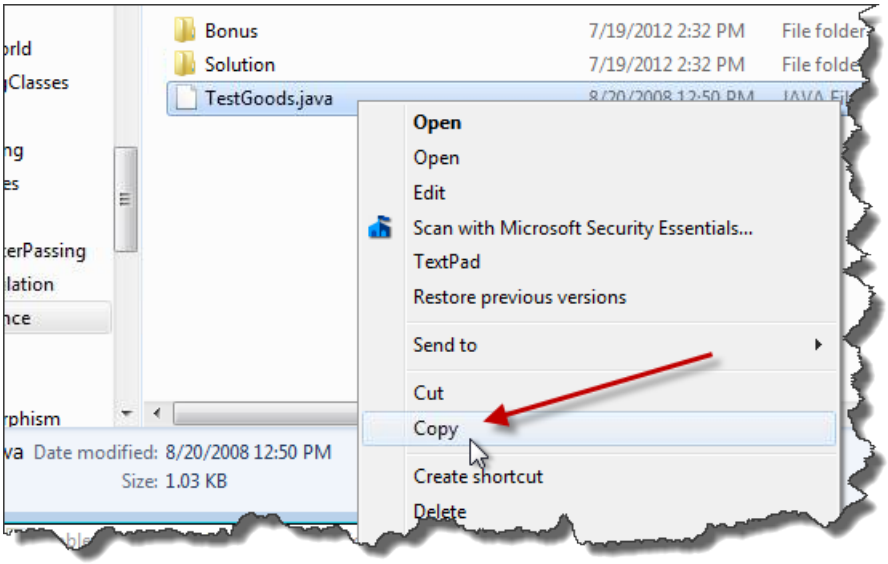
3.8 Save the Solid.java file once you have finished coding, and ensure there are no compiler errors.

Step 4: Run TestGoods

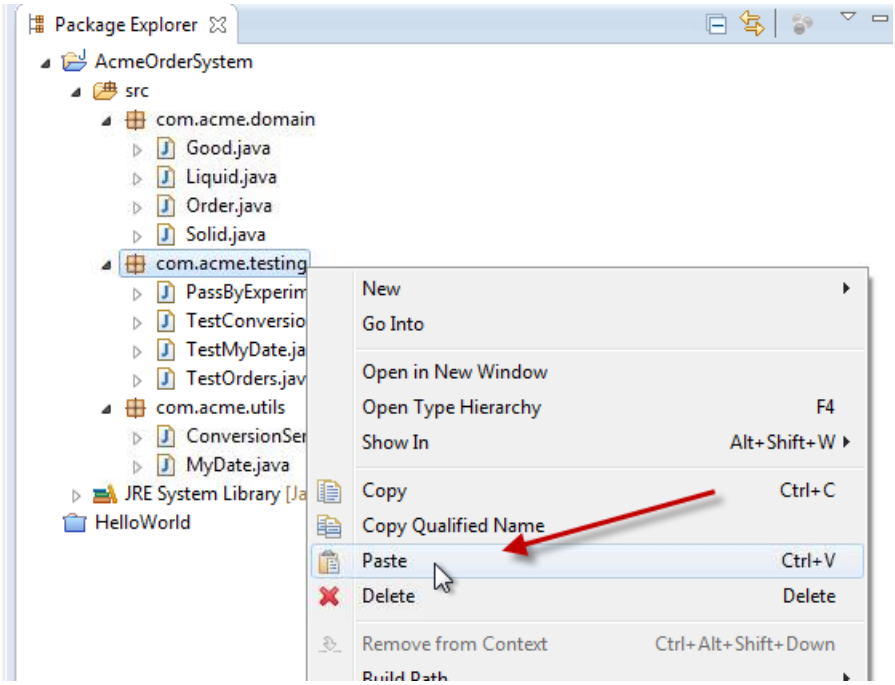
A program is provided to test your new classes. You don't need to write this program, as it is available in the lab folder.

4.1 Import TestGoods.java from the IntertechLearnJava zip (found in the Resources section for the "HelloWorld Lab").

4.1.1 Right-click on the file and request a copy of the file.



4.1.2 Back in the Eclipse IDE, right-click on the com.acme.testing package in the AcmeOrderSystem project and request to paste the copied file into the project.

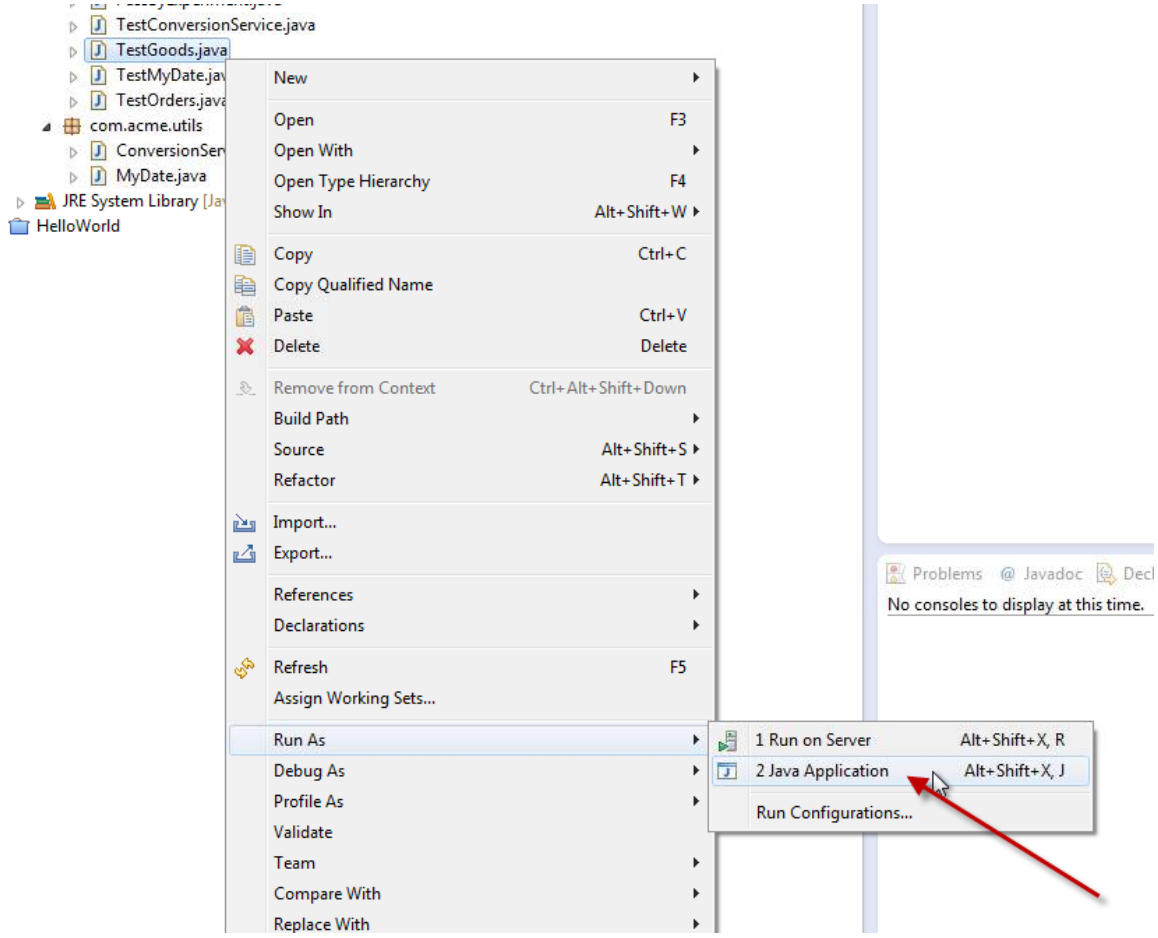


4.2 Open and explore TestGoods.java. Notice the creation of three "Goods." How is the reference variable "x" defined? Is this right? What exactly is x pointing to? The next chapter will have more on this concept.

4.3 Fix any errors in the domain classes or the TestGoods file.

4.4 Run the TestGoods class.

4.4.1 Right-click on TestGoods and select Run As > Java Application.



4.4.2 When you run the test, the output in the Console view should look like the following. If your run looks different, go back and try to fix it. Ask for help if you get stuck.

```
Acme Glue-2334 (liquid) 452.3893421169302 LITER
Acme Invisible Paint-2490 (liquid) 294.05307237600465 GALLON
Acme Anvil-1668 that is 0.075 CUBIC_METER in size
The weight of Acme Glue-2334 (liquid) 452.3893421169302 LITER is
6785.840131753953
The weight of Acme Invisible Paint-2490 (liquid)
294.05307237600465 GALLON is 205.83715066320323
```

```
The weight of Acme Anvil-1668 that is 0.075 CUBIC_METER in size
is 375.0
Is Acme Glue-2334 (liquid) 452.3893421169302 LITER flammable?
false
Is Acme Invisible Paint-2490 (liquid) 294.05307237600465 GALLON
flammable?  true
```

Lab Solutions

Good.java

```
package com.acme.domain;

public class Good {
  public enum UnitOfMeasureType {
    LITER, GALLON, CUBIC_METER, CUBIC_FEET
  }

  private String name;
  private int modelNumber;
  private double height;
  private UnitOfMeasureType unitOfMeasure;
  private boolean flammable = true;
  private double weightPerUofM;

  public Good(String name, int modelNumber, double height,
    UnitOfMeasureType uoM,
    boolean flammable, double wgtPerUoM) {
    this.name = name;
    this.modelNumber = modelNumber;
    this.height = height;
    this.unitOfMeasure = uoM;
    this.flammable = flammable;
    this.weightPerUofM = wgtPerUoM;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getModelNumber() {
    return modelNumber;
  }

  public void setModelNumber(int modelNumber) {
    this.modelNumber = modelNumber;
  }

  public double getHeight() {
    return height;
  }
```

```
  public void setHeight(double height) {
    this.height = height;
  }

  public UnitOfMeasureType getUnitOfMeasure() {
    return unitOfMeasure;
  }

  public void setUnitOfMeasure(UnitOfMeasureType unitOfMeasure)
{
    this.unitOfMeasure = unitOfMeasure;
  }

  public boolean isFlammable() {
    return flammable;
  }

  public void setFlammable(boolean flammable) {
    this.flammable = flammable;
  }

  public double getWeightPerUofM() {
    return weightPerUofM;
  }

  public void setWeightPerUofM(double weightPerUofM) {
    this.weightPerUofM = weightPerUofM;
  }

  public String toString() {
    return name + "-" + modelNumber;
  }

  public double volume() {
    return 0.0;
  }

  public double weight() {
    return volume() * weightPerUofM;
  }
}
```

Liquid.java

```
package com.acme.domain;

public class Liquid extends Good {
  private double radius;

  public Liquid(String name, int modelNumber, double height,
     UnitOfMeasureType uoM, boolean flammable, double
wgtPerUofM,
     double radius) {
        super(name, modelNumber, height, uoM, flammable,
wgtPerUofM);
        this.radius = radius;
  }

  public double getRadius() {
    return radius;
  }

  public void setRadius(double radius) {
    this.radius = radius;
  }

  public double volume() {
    return Math.PI * radius * radius * getHeight();
  }

  public String toString() {
    return super.toString() + " (liquid) " + volume() + " "
        + getUnitOfMeasure();
  }
}
```

Solid.java

```
package com.acme.domain;

public class Solid extends Good {
  private double width;
  private double length;

  public Solid(String name, int modelNumber, double height,
    UnitOfMeasureType uoM, boolean flammable, double wgtPerUofM,
    double width, double length) {
       super(name, modelNumber, height, uoM, flammable,
wgtPerUofM);
       this.width = width;
       this.length = length;
}

  public double getWidth() {
    return width;
  }

  public void setWidth(double width) {
    this.width = width;
  }

  public double getLength() {
    return length;
  }

  public void setLength(double length) {
    this.length = length;
  }

  public String toString() {
    return super.toString() + " that is " + volume() + " "
        + getUnitOfMeasure() + " in size";
  }

  public double volume() {
    return width * length * getHeight();
  }
}
```

TestGoods.java

```
package com.acme.testing;

import com.acme.domain.Good;
import com.acme.domain.Liquid;
import com.acme.domain.Solid;
import com.acme.domain.Good.UnitOfMeasureType;

public class TestGoods {

  public static void main(String[] args) {
    Liquid glue = new Liquid("Acme Glue", 2334, 4,
      UnitOfMeasureType.LITER, false, 15, 6);
    Liquid paint = new Liquid("Acme Invisible Paint", 2490,
0.65,
      UnitOfMeasureType.GALLON, true, 0.70, 12);
    Solid anvil = new Solid("Acme Anvil", 1668, 0.3,
      UnitOfMeasureType.CUBIC_METER, false, 5000, 0.5, 0.5);
    System.out.println(glue);
    System.out.println(paint);
    System.out.println(anvil);

    System.out.println("The weight of " + glue + " is " +
      glue.weight());
    System.out.println("The weight of " + paint + " is " +
      paint.weight());
    System.out.println("The weight of " + anvil + " is " +
      anvil.weight());

    Good x = glue;
    System.out.println("Is " + x + " flammable?  " +
x.isFlammable());
    x = paint;
    System.out.println("Is " + x + " flammable?  " +
x.isFlammable());
  }
}
```

Bonus Lab

Step 5: Add a "final" method

Final on a class makes it non-extendable. Final on a variable makes it a constant (like PI on the Math class). Final on a method protects it from being overridden. In this step, add a final method to the Good class.

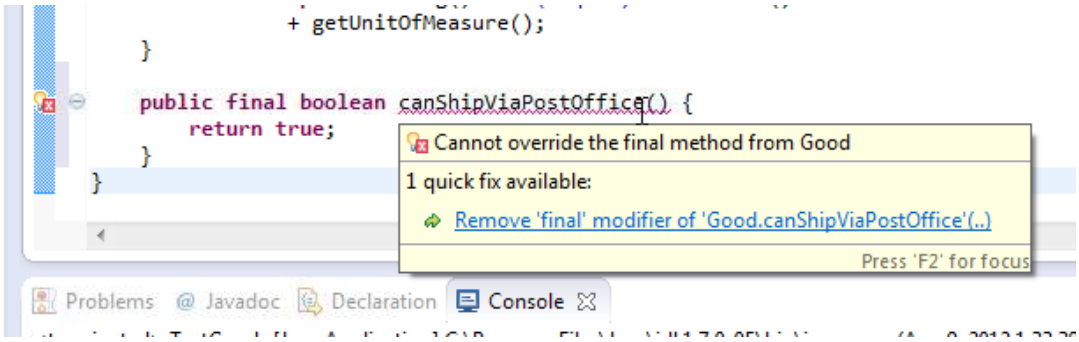
5.1 Add canShipViaPostOffice( ) to the Good class and make it final. The canShipViaPostOffice( ) method should return a boolean indicating whether the Good can be shipped via the Post Office. The method should return true if the Good is not flammable and if the weight < 200. Make the method final so that it cannot be overridden in either Solid or Liquid classes.

```
public final boolean canShipViaPostOffice() {
  //...your code here
}
```

5.2 Add calls in TestGoods.java to check your work.

```
System.out.println(glue + " can ship via Post office?" +
glue.canShipViaPostOffice());
System.out.println(anvil + " can ship via Post office?" +
anvil.canShipViaPostOffice());
System.out.println(paint + " can ship via Post office?" +
paint.canShipViaPostOffice());
```

5.3 Try to add a canShipViaPostOffice( ) method to Solid or Liquid. Do you get a compiler error?



Give Intertech a call at **1.800.866.9884** or visit Intertech's website.