Lab Exercise: More with Collections

INTERTECH

Instructors Who Consult. | **Consultants** Who Teach.

About Intertech

Thank you for choosing Intertech for your training from Udemy. 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 such as the course in which you've enrolled from Udemy. 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**.

As a Udemy customer, you can save on any live in-classroom, live online, or onsite training with Intertech as well. Just use promo code "Udemy_Labs" when enrolling **and** save 35% on the course price.

We appreciate you choosing Intertech on Udemy for this course!

Lab Exercise

More with Collections

With a better understanding of the Collections Framework, you can now add better type safety, where warranted, to your application via Generics. Additionally, you have learned about some convenience capability to allow you to sort/search your collections.

In this lab, you explore improve your existing code to remove type warnings and add the means to sort and search your catalog of goods.

Specifically, in this lab you will:

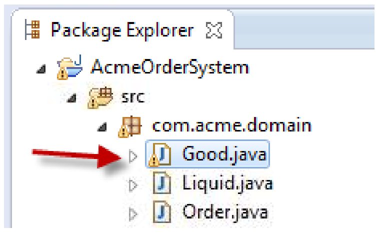
- Use generics to type your collection instances.
- Sort the catalog of goods.
- Optionally try out other collection operations, like search, on your catalog of goods.

Scenario

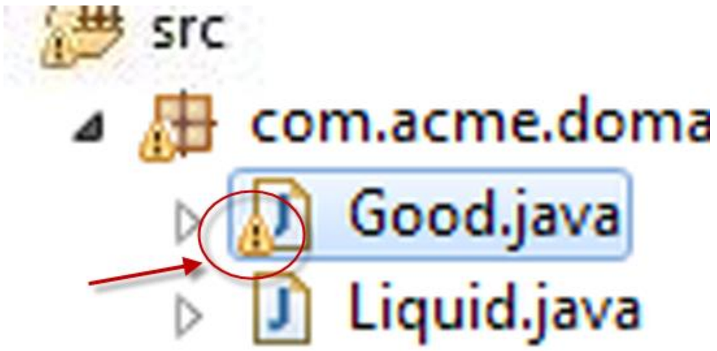
It's time now to remove those compiler warnings from your last lab.

Step 1: Examine the compiler warnings.

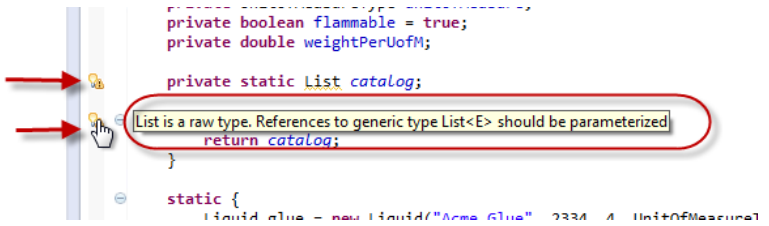
1.1 Open Good.java and examine the compiler warnings that result from the loose typing associated to the collections you use. In the Package Explorer view, double-click on the Good.java file in the com.acme.domain package to open the file in a Java editor.



Note: You might notice the yellow warning sign next to Good.java and TestGoods.java file (as highlighted in the picture below. This is an indication of a warning not an error. While compiler warnings still allow code to execute, it is always a good idea to study compiler warnings and understand what is causing them before disregarding them.



1.2 Take note of all the compiler warnings in this class as a result of not being type specific with regard to the list and set objects in the code.



```
    private boolean flammable = true;
    private double weightPerUofM;

    private static List catalog;

List is a raw type. References to generic type List<E> should be parameterized
        return catalog;
    }

    static {
```

Step 2: Modify Good.java to use Generics

2.1 Modify the type declaration on the catalog static variable to specify the intended data type, which is Good, of the contents of the catalog list as shown below.

```
private static List<Good> catalog;
```

2.2 Modify return types to use specific data types. Make a similar change to the return type of the getCatalog( ) method.

```
public static List<Good> getCatalog() {
  return catalog;
}
```

2.3 When creating instances of a collection, also provide the more specific content type. In the static initialization block of the Good class, create an ArrayList of Good using generics.

```
...
catalog = new ArrayList<Good>();
...
```

2.4 Modify the flammablesList( ) method to use generics. There are several data typing opportunities in the flammablesList( ) method. The return type, local variable definition and even the iterator can all be made more specific with generics. Also, note that getting the next item from the iterator no longer requires a cast be done (since Java already knows the type of object in the set)!

```
public static Set<Good> flammablesList() {
  Set<Good> flammables = new HashSet<Good>();
  Iterator<Good> i = Good.getCatalog().iterator();
  while (i.hasNext()) {
    Good x = i.next();
    if (x.isFlammable()) {
      flammables.add(x);
    }
  }
  return flammables;
}
```

2.5 Save and test. Save your changes to the Good class. Make sure there are no compiler errors in the class. Run TestGoods and check the output. The output should not have been affected by the use of generics.

```
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
[Acme Glue-2334 (liquid) 452.3893421169302 LITER, Acme Invisible
Paint-2490 (liquid) 294.05307237600465 GALLON, Acme Anvil-1668
that is 0.0225 CUBIC_METER in size, Acme Safe-1672 that is 0.25
CUBIC_METER in size, Acme Balloon-1401 that is 375.0 CUBIC_FEET
in size, Acme Disintegrating Pistol-1587 that is 0.1 CUBIC_FEET
in size, Acme Nitroglycerin-4289 (liquid) 7.0685834705770345
CUBIC_METER, Acme Oil-4275 (liquid) 7.0685834705770345
CUBIC_METER]
[Acme Glue-2334 (liquid) 452.3893421169302 LITER, Acme Anvil-
1668 that is 0.0225 CUBIC_METER in size, Acme Safe-1672 that is
0.25 CUBIC_METER in size, Acme Balloon-1401 that is 375.0
CUBIC_FEET in size, Acme Disintegrating Pistol-1587 that is 0.1
CUBIC_FEET in size, Acme Nitroglycerin-4289 (liquid)
7.0685834705770345 CUBIC_METER, Acme Oil-4275 (liquid)
7.0685834705770345 CUBIC_METER, Acme Toaster-1755 that is 0.75
CUBIC_FEET in size, Acme Toaster-1755 that is 0.75 CUBIC_FEET in
size]
Flammable products:  [Acme Oil-4275 (liquid) 7.0685834705770345
CUBIC_METER, Acme Nitroglycerin-4289 (liquid) 7.0685834705770345
CUBIC_METER]
```

Note: Importantly, however, notice that once Good.java is saved and compiles correctly, the compiler warnings disappear on both Good.java as well as TestGoods.java.

Step 3: Order the catalog

The catalog has the correct goods in it, but the order of the information displayed is dependent on how the goods were added to the catalog. The business wants the catalog sorted by the name of the good. Use the Comparable interface to implement this change.

3.1 Change Good to implement Comparable. Open Good.java in a Java editor. Change the class declaration to have Good implement the java.lang.Comparable interface.

```
public abstract class Good implements Product, Comparable<Good>
{
...
}
```

Note: even the Comparable interface works on generics – in this case indicating you intend to implement the comparison of Good objects.

3.2 Add the compareTo( ) method. The Comparable interface requires the implementing class to implement the compareTo( ) method. Add the compareTo() method as shown below to Good.java.

```
public int compareTo(Good o) {
  return getName().compareTo(o.getName());
}
```

Note: Can you tell how goods will be ordered given this method? After you get the application working, come back to this method and see if you can change the catalog to sort by model number.

3.3 At the bottom of the main( ) method in TestGoods.java, use the java.utils.Collections class to sort the catalog ArrayList. Display the catalog one more time after sorting.

```
Collections.sort((List<Good>) Good.getCatalog());
System.out.println(Good.getCatalog());
```

3.4 Import the necessary classes. The last two lines in the main( ) method of TestGoods.java require you add some imports to TestGoods.java.

```
import java.util.List;
import java.util.Collections;
```

3.5 Save your changes to the Good and TestGoods classes. Make sure there are no compiler errors in either class. Run TestGoods and check the output. Now the list of goods should be sorted by name.

```
...
[Acme Anvil-1668 that is 0.0225 CUBIC_METER in size, Acme
Balloon-1401 that is 375.0 CUBIC_FEET in size, Acme
Disintegrating Pistol-1587 that is 0.1 CUBIC_FEET in size, Acme
Glue-2334 (liquid) 452.3893421169302 LITER, Acme Nitroglycerin-
4289 (liquid) 7.0685834705770345 CUBIC_METER, Acme Oil-4275
(liquid) 7.0685834705770345 CUBIC_METER, Acme Safe-1672 that is
0.25 CUBIC_METER in size, Acme Toaster-1755 that is 0.75
CUBIC_FEET in size, Acme Toaster-1755 that is 0.75 CUBIC_FEET in
size]
```

Lab Solutions

Good.java

```
package com.acme.domain;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

public abstract class Good implements Product, Comparable<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;

  private static List<Good> catalog;

  public static List<Good> getCatalog() {
    return catalog;
  }

  static {
    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, 500, 0.25, 0.3);
    Solid safe = new Solid("Acme Safe", 1672, 1.0,
        UnitOfMeasureType.CUBIC_METER, false, 300, 0.5, 0.5);
    Solid balloon = new Solid("Acme Balloon", 1401, 15,
        UnitOfMeasureType.CUBIC_FEET, false, 10, 5, 5);
    Solid pistol = new Solid("Acme Disintegrating Pistol", 1587,
0.1,
        UnitOfMeasureType.CUBIC_FEET, false, 1, 0.5, 2);
    Liquid nitro = new Liquid("Acme Nitroglycerin", 4289, 1.0,
        UnitOfMeasureType.CUBIC_METER, true, 0.25, 1.5);
    Liquid oil = new Liquid("Acme Oil", 4275, 1.0,
        UnitOfMeasureType.CUBIC_METER, true, 0.25, 1.5);
    catalog = new ArrayList<Good>();
    catalog.add(glue);
    catalog.add(paint);
    catalog.add(anvil);
    catalog.add(safe);
    catalog.add(balloon);
    catalog.add(pistol);
    catalog.add(nitro);
    catalog.add(oil);
  }

  public static Set<Good> flammablesList() {
    Set<Good> flammables = new HashSet<Good>();
    Iterator<Good> i = Good.getCatalog().iterator();
    while (i.hasNext()) {
      Good x = i.next();
      if (x.isFlammable()) {
        flammables.add(x);
      }
    }
    return flammables;
  }

  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 abstract double volume();

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

  public int compareTo(Good o) {
    // to order by model number, use the following
    // lines of code instead
    // if (getModelNumber() < o.getModelNumber()) {
```

```
    // return -1;
    // } else if (getModelNumber() > o.getModelNumber()) {
    // return 1;
    // } else return 0;
    return getName().compareTo(o.getName());
  }
}
```

TestGoods.java

```
package com.acme.testing;

import java.util.Collections;
import java.util.List;
import com.acme.domain.Good;
import com.acme.domain.Good.UnitOfMeasureType;
import com.acme.domain.Liquid;
import com.acme.domain.Solid;

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());

    System.out.println(Good.getCatalog());
    Good.getCatalog().remove(1);
    Solid toaster = new Solid("Acme Toaster", 1755, 0.75,
        UnitOfMeasureType.CUBIC_FEET, false, 1.0, 1.0, 1.0);
    Good.getCatalog().add(toaster);
    Good.getCatalog().add(toaster);
    System.out.println(Good.getCatalog());
    System.out.println("Flammable products:  " +
      Good.flammablesList());

    Collections.sort((List<Good>) Good.getCatalog());
    System.out.println(Good.getCatalog());
  }
}
```

Bonus Lab

Step 4: Search for a good

4.1 In the TestGoods class, use the Collections class to search the product catalog for a previously defined good like glue. Make sure you perform the search after sorting the catalog or else the search won't work properly. The results of your search should be displayed similar to the results highlighted below.

```
[Acme Anvil-1668 that is 0.0225 CUBIC_METER in size, Acme
Balloon-1401 that is 375.0 CUBIC_FEET in size, Acme
Disintegrating Pistol-1587 that is 0.1 CUBIC_FEET in size, Acme
Glue-2334 (liquid) 452.3893421169302 LITER, Acme Nitroglycerin-
4289 (liquid) 7.0685834705770345 CUBIC_METER, Acme Oil-4275
(liquid) 7.0685834705770345 CUBIC_METER, Acme Safe-1672 that is
0.25 CUBIC_METER in size, Acme Toaster-1755 that is 0.75
CUBIC_FEET in size, Acme Toaster-1755 that is 0.75 CUBIC_FEET in
size]
Found Acme Invisible Paint-2490 (liquid) 294.05307237600465
GALLON in the catalog at location: 3
```

4.2 What happens when you search for a good no longer in the catalog (like paint)? Can you explain the results (check the Javadocs on Collections for help)?

```
Found Acme Invisible Paint-2490 (liquid) 294.05307237600465
GALLON in the catalog at location: -5
```

Step 5: HashSet in place of ArrayList

The benefit of using many of the Collection classes is that they are largely interchangeable. As part of the bonus lab in the last chapter, you used HashSet instead of an ArrayList. What part of the code breaks now if you try to use HashSet in place of ArrayList? Why is this?

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