

Purpose	<ol style="list-style-type: none"> 1. Use an <i>abstract superclass</i> to implement common behavior and eliminate duplicate code. 2. See how one interface can extend another. 3. Practice refactoring in Eclipse. 4. Revise the design to include a currency for money.
What to Submit	<p>Commit your code to your "coinpurse" project on Bitbucket.</p> <ol style="list-style-type: none"> 1. Before committing your work, create an annotated tag named "LAB3" for your Lab3 coin purse. 2. Commit your work to the same project. 3. Add an annotated tag "LAB4" to indicate this revision.

In a previous lab, you wrote an interface for *Valuable* objects to enable polymorphism, and defined several kinds of valuable objects that can be put in a Purse.

This makes the Purse a lot more flexible, but results in some duplicate code.

For example, `equals()` and `getValue()` are the same in most or all the classes implementing *Valuable*.

1. Create an Abstract Superclass for Valuables

To eliminate duplicate code, create an *Abstract Superclass* named **AbstractValuable**.

Note: If you are using Eclipse, then you can do 1.1 - 1.3 using *Refactoring*. See below for how.

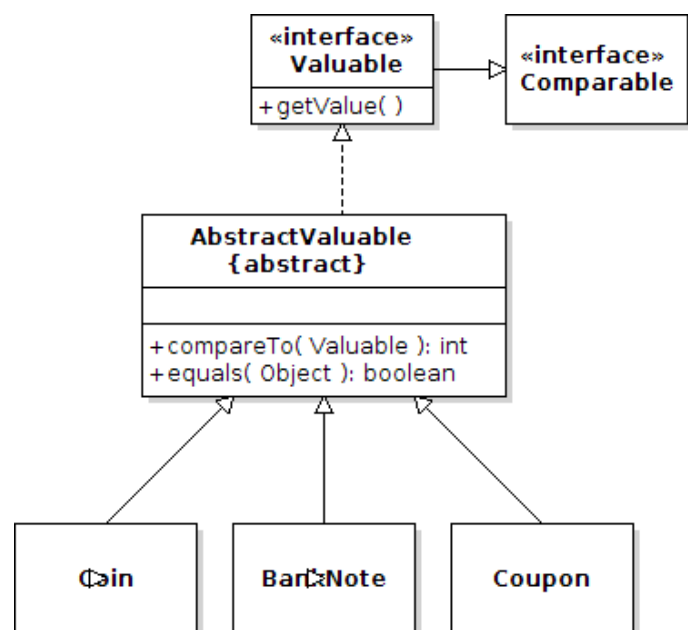
1.1 Declare the **AbstractValuable** class to be **abstract**.

1.2 Declare that **Coin**, **BankNote**, and **Coupon** are subclasses of **AbstractValuable**.

1.3 Move identical code for **equals()** to the superclass. If you are using Eclipse, use *Refactoring* to extract the methods.

Verify that **equals()** in **AbstractValuable** works correctly for each subclass. To achieve this, for testing that two objects belong to the same class use:

`if (this.getClass() == obj.getClass()) ...`



Note: in the `equals` and `compareTo` of **AbstractValuable**, you should use `getValue()` to get the value of objects, so that each object is free to determine its value any way it wants. *Don't* try to directly access the `value` attribute.

Why? A subclass may want to redefine how its value is computed. Using `getValue()` obeys the principle "*Program to an interface, not to an implementation.*"

2. Modify Valuable to extend Comparable

Every **Valuable** class *must* be comparable by value: a 10-Baht coin is worth less than a 50-Baht **BankNote**, which is worth less than a Red coupon. So it makes sense that *Valuable* itself should be *Comparable*.

2.1 Modify *Valuable* to declare it is also *Comparable*.

2.2 Delete "... implements *Valuable*" from **Coin**, **BankNote**, and **Coupon**.

2.3 Be careful how you do the comparison. We may have some kinds of *Valuable* with very small value (0.000001) or very large value (1,000,000,000).

3. What about the `value` attribute?

What about the `value` attribute in `Coin`, `BankNote`, and (maybe) `Coupon`?

Can you move `value` to the superclass? What should you do with `getValue()`?

4. Verify Your Code

Check your code for these properties:

- `Coin`, `BankNote`, and `Coupon` do not have `equals` or `compareTo`, they use the methods from *`AbstractValuable`*.
- The `Purse` and user interface do not depend on *`AbstractValuable`*. They depend only on `Valuable`.
- The money classes (`Coin`, `BankNote`, and `Coupon`) don't contain any duplicate code.

Refactoring in Eclipse

"*Refactoring*" means to restructure your source code. Eclipse and NetBeans have many refactoring operations to save time & reduce errors. The "Extract Superclass" refactoring creates a superclass and can move methods to the superclass.

We want to create an abstract superclass for **BankNote**, **Coin**, and **Coupon**.

Eclipse can create an **AbstractValuable** superclass for you. Follow these steps:

1. Open one of the classes in the Eclipse editor.

Double-click on **Coin** to edit it.

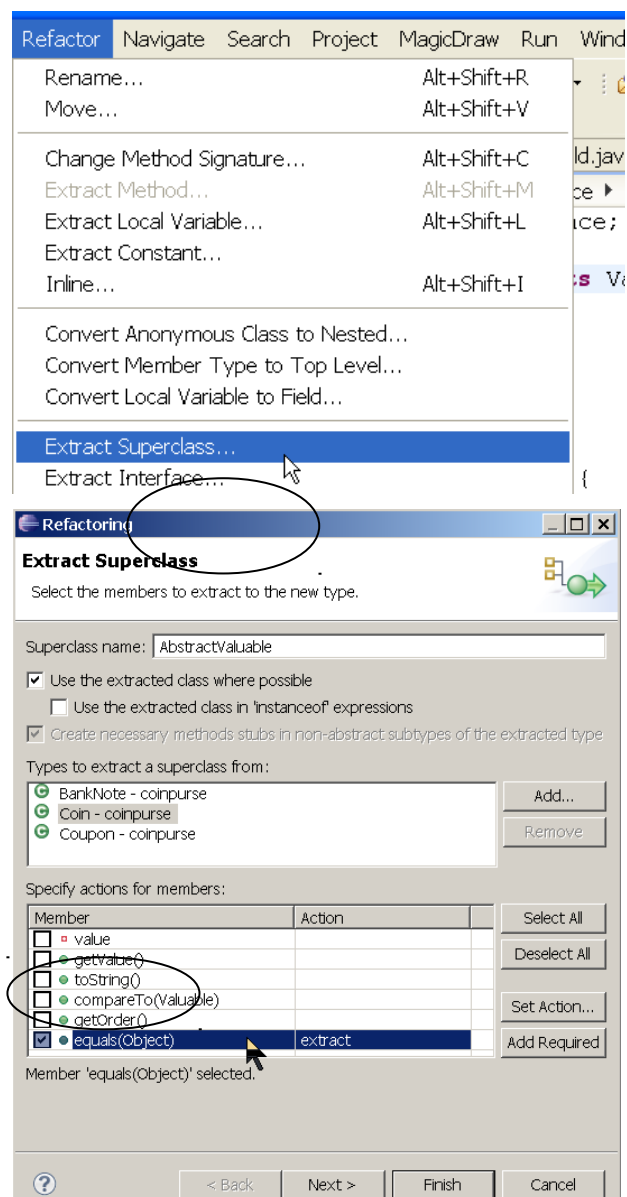
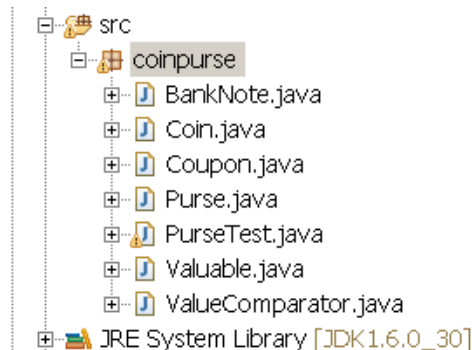
2. From the **Refactor** menu select **Extract Superclass...**

3. In the **Extract Superclass** dialog, enter **AbstractValuable** as the Superclass name.

4. Click **Add...** and add other classes you want to refactor (**BankNote** and **Coupon**).

5. Select **equals(Object)** as the method to extract to the superclass.

Click the **Next>** Button.

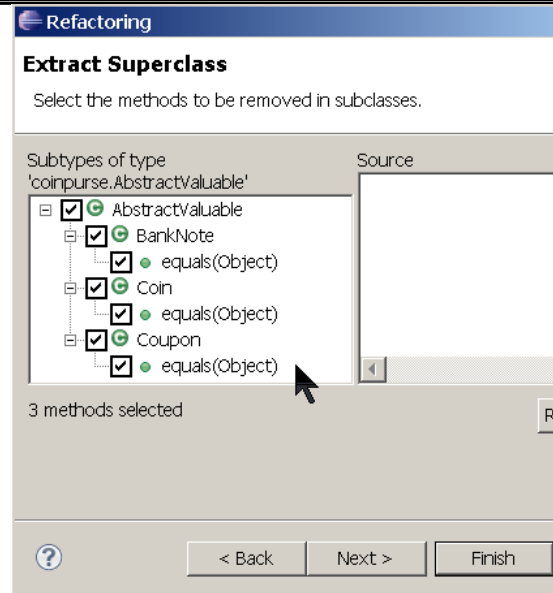


6. This dialog shows which classes will be subclasses and what methods will be extracted to AbstractValuable.

Check equals method for all 3 subclasses.

Click Finish.

You can ignore warning message about problems. You can fix these problems after refactoring.



How to Extract More Methods to Superclass

You can move other methods from a subclasses to a superclass . In the Refactor menu choose "Pull Up".

Undo Refactoring

If you make a mistake, you can Undo refactoring (Edit -> Undo or Refactor -> History).