Extracting superclass

Object Oriented Programming
2024 First Semester
Shin-chi Tadaki (Saga University)

1. Class Hierarchy in Java
2. Extracting superclass
3. Exercise: Selection Sort

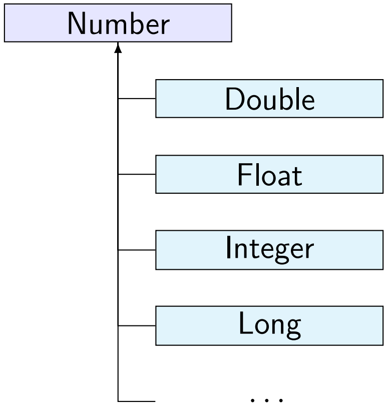
The *Object* class

- The `Object` class is a superclass of all other classes. in Java
- Methods of the `Object` class
  - clone(): creates a copy of this object.
  - equal(Object obj): returns True if `obj` is equal to this one.
  - getClass(): returns the runtime class of this one.
  - hashCode(): returns the hash code of this one.
  - notify(): wakes a single thread waiting this.
  - notifyAll(): wakes all threads waiting this.
  - toString(): returns a string representation of this.
  - wait(): causes the current thread to wait.

Example of the Class Hierarchy

- The `Number` class is a subclass of the `Object` class
  - It is an abstract super class of classes expressing numerals
  - It implements the `Serializable` interface
    - `Serializable` objects can be written to streams (files, networks, etc.).
- The `Integer` class is a subclass of the `Number` class
  - It implements the `Comparable<Integer>` interface

`Number` and its extensions



Extracting superclass

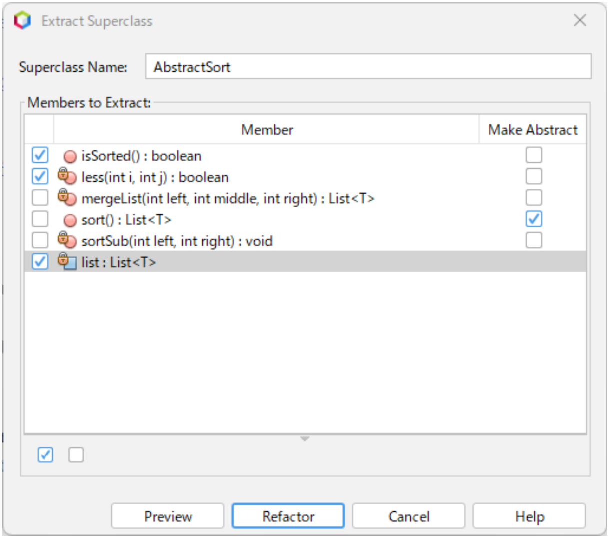
- Extracting common features from existing classes
- The *refactoring* function in NetBeans is available
  - *refactoring*: changing the structure of the code without changing its behavior
- Preparation
  - Create a new package `example2`.
  - Copy the followings from `example1` to `example2` with *refactoring*.
    - `BubbleSort`
    - `MergeSort`
  - Delete `import example1.*` in each source code.

Extracting features from `MergeSort`

- Select the menu `Refactor`→`Extract Superclass`
- Extract the followings with the current implementations `less()`, `isSorted()`, `list`
- Extract the following as `abstract` `sort()`
- Save as `AbstractSort`
- Confirm the constructor

See the next sheet.

Extract Superclass in NetBeans

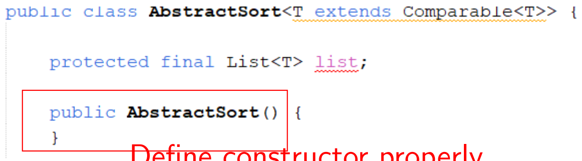

Extract Superclass

Superclass Name: AbstractSort

Members to Extract:

	Member	Make Abstract
✓	isSorted() : boolean	☐
✓	less(int i, int j) : boolean	☐
☐	mergeList(int left, int middle, int right) : List<T>	☐
☐	sort() : List<T>	✓
☐	sortSub(int left, int right) : void	☐
✓	list : List<T>	

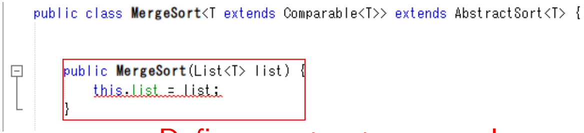
Modify `AbstractSort`



```
public class AbstractSort<T extends Comparable<T>> {

    protected final List<T> list;

    public AbstractSort() {
    }
```

Define constructor properly

Modify `MergeSort`



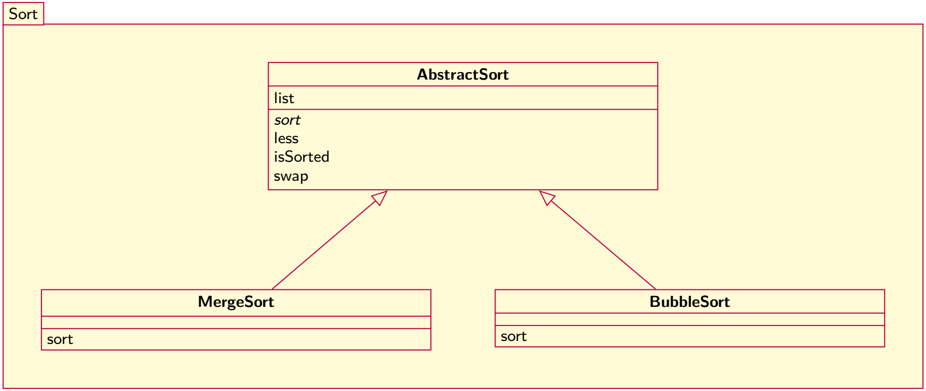
```
public class MergeSort<T extends Comparable<T>> extends AbstractSort<T> {

    public MergeSort(List<T> list) {
        this.list = list;
    }
```

Define constructor properly

Subclasses of `AbstractSort`

- `MergeSort`
- `BubbleSort`
- These subclasses override the `sort()`

Class Inheritance



Exercise: Redefine `BubbleSort`

- Redefine `BubbleSort` as a subclass of `AbstractSort`
- Define constructor properly
- Implements `sort()` method
- Delete common (inherited) methods

Exercise: Selection Sort

Algorithm 1 Selection Sort for list $d_i (0 \leq i < n)$

```
for $i = 0; i < n - 1; i++$ do
    $m$ is the index of the minimum element after $i$
    if $m \neq i$ then
        swap($i, m$)
    end if
end for
```

Exercise

- Define `SelectionSort` class as a subclass of `AbstractSort`.
- Define `protected void swap(int,int)` in `AbstractSort`.
- And confirm it work.