

# Software 1

Recitations No. 7/8  
(More on Contracts, Refactoring)

# BoundedVersionedString

- A VersionedString with a bounded capacity
- Abstraction Function:

$$A : BVS \rightarrow \{\phi\} \cup S \cup S \times S \cup \dots \cup S^{capacity}$$

where:

- $BVS = \{vs \mid vs \text{ is a BoundedVersionedString state}\}$
- $S = \{ s \mid s \text{ is a String}\}$

$$vs \in BVS \xrightarrow{A} \begin{cases} \phi \\ s \in S \\ (s_1, s_2) \in S \times S \\ (s_1, s_2, s_3) \in S \times S \times S \end{cases}$$

# Method Specification

---

```
public class BoundedVersionedString {  
    public BoundedVersionedString(int capacity) {...}  
    public void add(String s) {...}  
    public int length() {...}  
    public String getLastVersion() {...}  
    public String getVersion(int i) {...}  
}
```

# The Contract

## Pre/Post Conditions

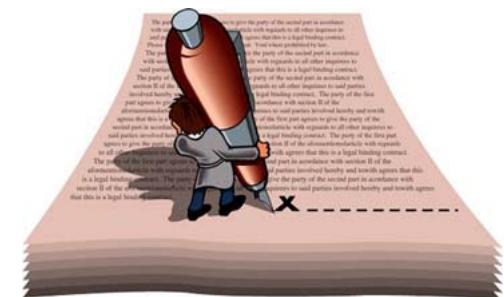
### ■ BoundedVersionedString(int capacity):

- Requires:

- $\text{capacity} > 0$

- Ensures:

- nothing



# The Contract

## Pre/Post Conditions (cont.)

■ void add(String s):

- Requires:

s != null

- Ensures:

$$A(\text{old}) = \phi \Rightarrow A(\text{new}) = (s)$$

$$A(\text{old}) = (s_1, s_2, \dots, s_{k \geq 1}) \Rightarrow A(\text{new}) = \begin{cases} (s_1, s_2, \dots, s_k, s) & \text{if } k < \text{capacity} \\ (s_2, s_3, \dots, s_k, s) & \text{if } k = \text{capacity} \end{cases}$$

# The Contract

## Pre/Post Conditions (cont.)

■ int length():

- Requires:

nothing

- Ensures:

$$A(new) = A(old)$$

Since length() is a query

$$\text{returned value} = \begin{cases} 0 & \text{if } A(old) = \phi \\ k & \text{if } A(old) = (s_1, \dots, s_{k \geq 1}) \end{cases}$$

# The Contract

## Pre/Post Conditions (cont.)

■ String getVersion(int i):

- Requires:

$$A(this) = (s_1, s_2, \dots, s_{k \geq i \geq 1})$$

- Ensures:

$$A(new) = A(old)$$

$$A(old) = (s_1, \dots, s_{k \geq i \geq 1}) \Rightarrow \text{returned value} = s_i$$

# The Contract

## Pre/Post Conditions (cont.)

■ String getLastVersion():

- Requires:

$$A(this) \neq \phi$$

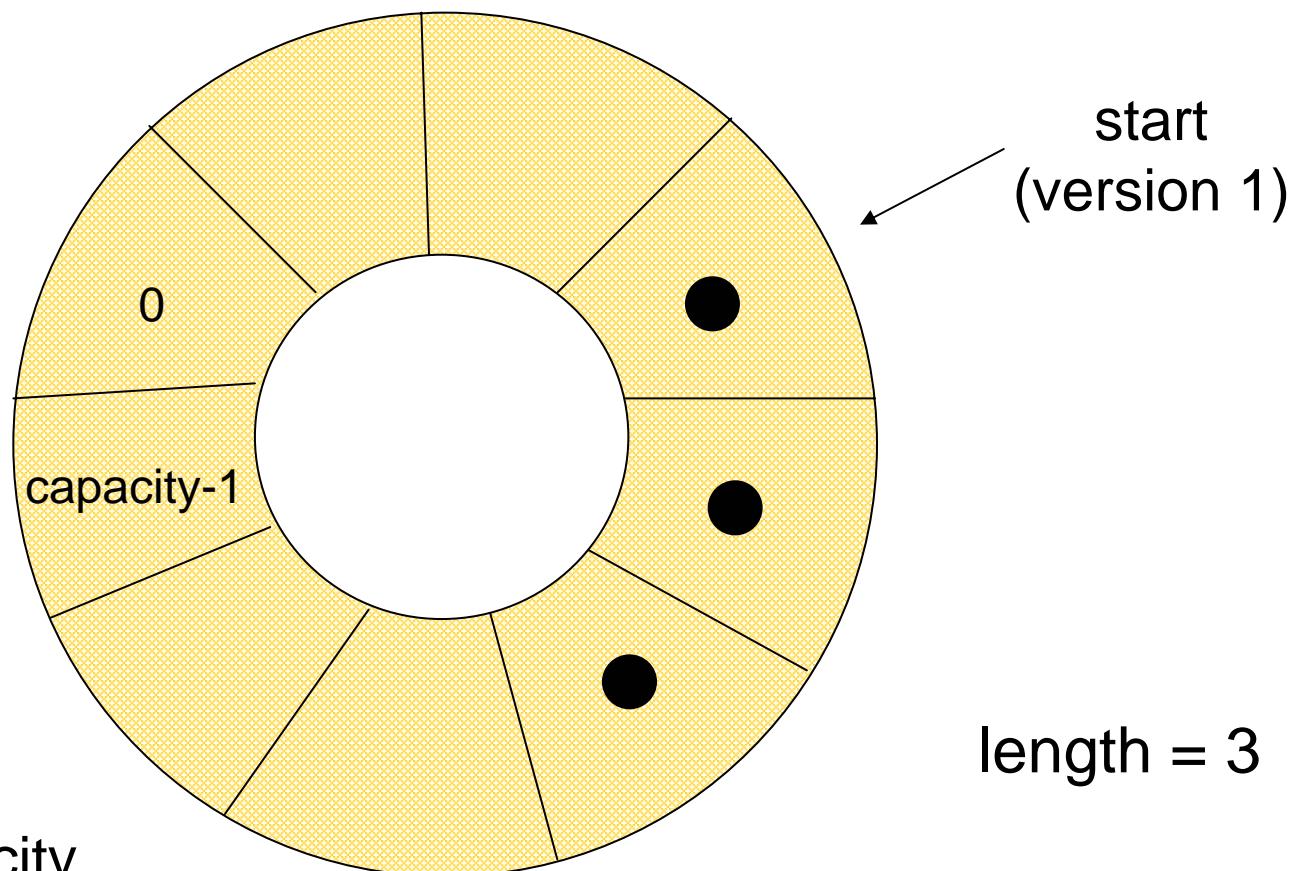
- Ensures:

$$A(new) = A(old)$$

$$A(old) = (s_1, \dots, s_k) \Rightarrow \text{returned value} = s_k$$

# Implementation

- Based on a circular array



Array size = capacity

# Implementation (cont.)

## Fields

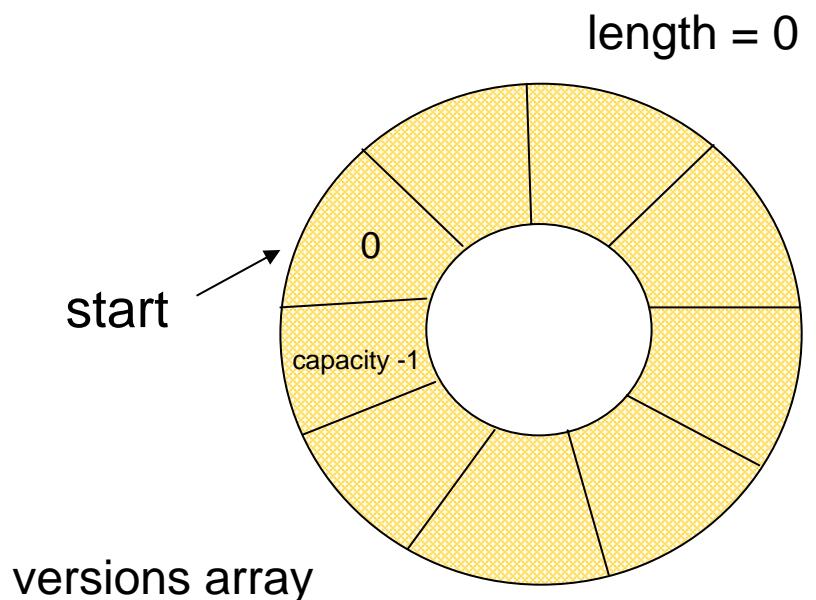
---

```
// Stores the remembered versions
private String[] versions ;  
  
// The maximum number of versions to remember
private int capacity;  
  
// The actual number of remembered versions
private int length = 0;  
  
// The position of the oldest remembered version
private int start = 0;
```

# Implementation (cont.)

## Constructor

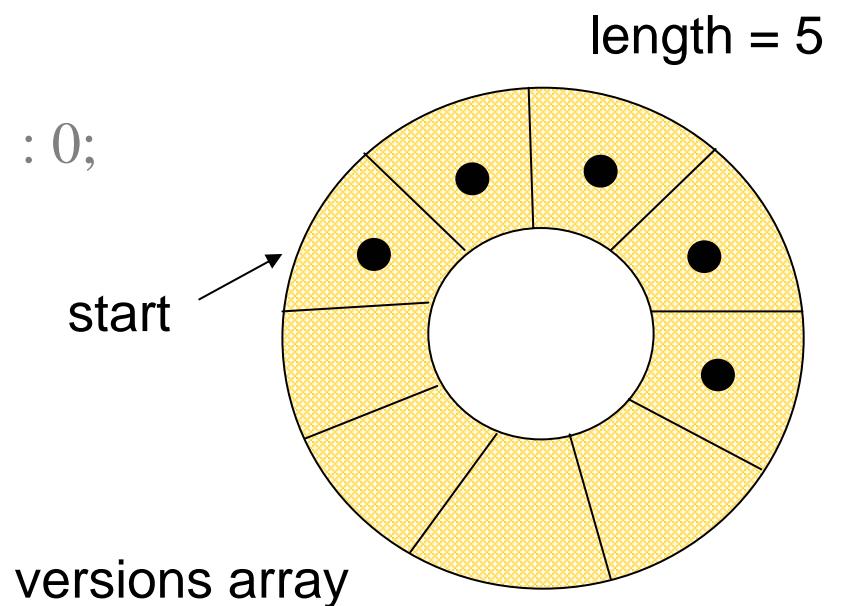
```
public BoundedVersionString(int capacity) {  
    this.capacity = capacity;  
    versions = new String[capacity];  
}
```



# Implementation (cont.)

## Command

```
public void add(String s) {  
    int pos = (start + length) % capacity;  
    versions[pos] = s;  
  
    if ((length > 0) && (start == pos)) {  
        start = (start+1) % capacity;  
    }  
  
    length += (length < capacity) ? 1 : 0;  
}
```



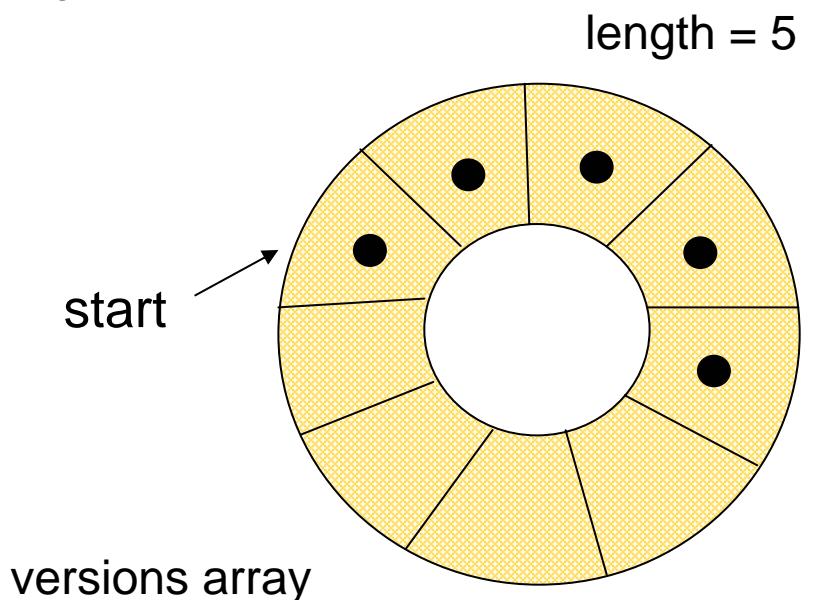
# Implementation (cont.)

## Queries

```
public int length() {  
    return length;  
}
```

```
public String getVersion(int i) {  
    return versions[(start+i-1)%capacity]);  
}
```

```
public String getLastVersion() {  
    return getVersion(length());  
}
```



# Abstraction Function Definition

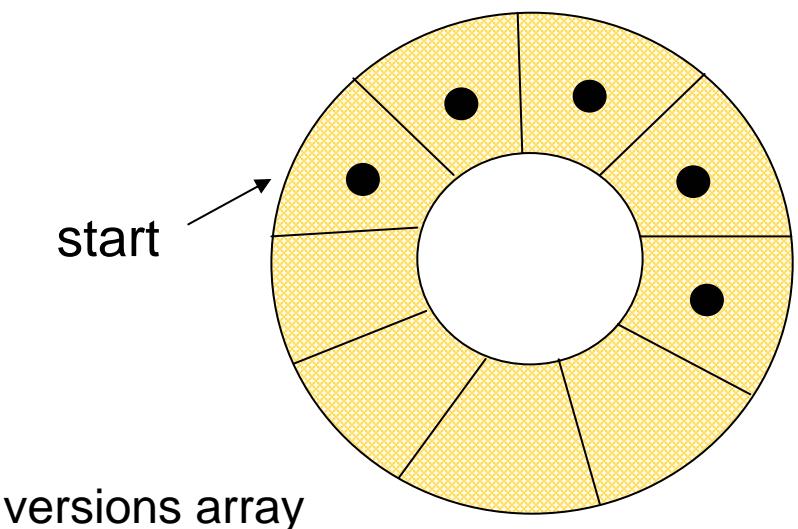
$$A : BVS \rightarrow \{\phi\} \cup S \cup S \times S \cup \dots \cup S^{capacity}$$

where:

- $BVS = \{vs \mid vs \text{ is a BoundedVersionedString state}\}$
- $S = \{ s \mid s \text{ is a String}\}$

---

$$A(this) = (versions [start], \dots, versions [(start + length - 1) \% capacity])$$



# Representation Invariant

---

1.  $capacity > 0$
2.  $0 \leq start < capacity$
3.  $length < capacity \Rightarrow start = 0$
4.  $length = \begin{cases} 0 & \text{if } A(this) = \phi \\ k & \text{if } A(this) = (s_1, \dots, s_{k \geq 1}) \end{cases}$

# Refactoring

---

- Changing the structure of a software without changing its functionality
- Comes from the *factorization* math term:
  - $x^2-a^2 = (x-a) \cdot (x+a)$
- A kind of code reorganization
- Not code rewriting
- Done in small steps with testing
- An aspect of extreme programming

# Eclipse Refactoring

---

- An automated tool:

- More efficient
- Reduce the risk of introducing bugs

- Refactoring types:

- Physical Organization
- Logical Organization
- Restructure within a class

# Eclipse Refactoring

## Physical Reorganization

---

- Rename variables, fields, methods, classes, interfaces, packages etc.
- Move packages, classes, methods etc.
- Change Method Signature
- Convert Anonymous Class to Nested
- Move Member Type to New File

# Eclipse Refactoring

## Logical Reorganization

- Push Down
- Pull Up
- Extract Interface
- Generalize Type
- Use Supertype Where Possible

# Eclipse Refactoring

## Restructure within a class

- Inline
- Extract Method
- Extract Local Variable
- Extract Constant
- Introduce Parameter
- Introduce Factory
- Convert Local Variable to Field
- Encapsulated Fields

# Eclipse Refactoring

## Demo

Initial Stage:

class  
VersionedString

class  
BoundedVersionedString

Stage 1:

interface  
VersionedString

class  
LinkedVersionedString

class  
BoundedVersionedString

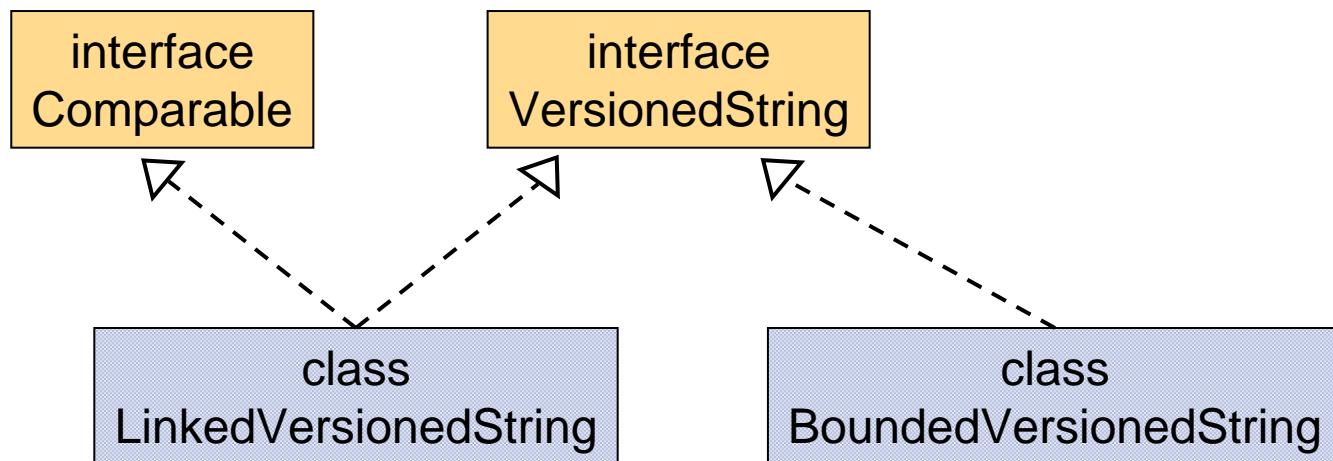


# Eclipse Refactoring

## Demo (cont.)

### Stage 2:

Support sorting `LinkedVersionedString` objects:



# Eclipse Refactoring

## Demo (cont.)

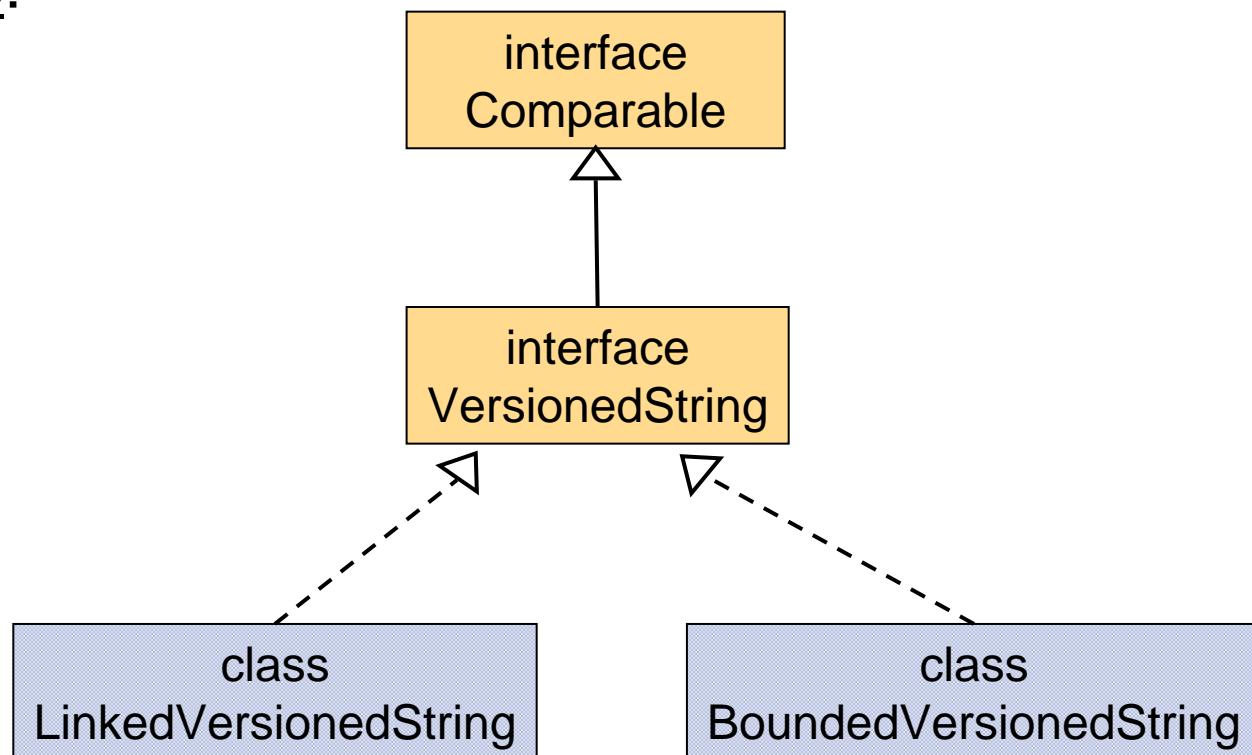
### Stage 2 (cont.):

```
public class LinkedVersionedString implements VersionString, Comparable {  
    ....  
    public int compareTo(Comparable other) throws IncomparableException {  
        LinkedVersionedString other_vs;  
        try {  
            other_vs = (LinkedVersionedString) other;  
        } catch (java.lang.ClassCastException ce) {  
            throw new IncomparableException();  
        }  
  
        if (this.length() > other_vs.length()) return 1;  
        if (this.length() < other_vs.length()) return -1;  
        return 0;  
    }  
}
```

# Eclipse Refactoring

## Demo (cont.)

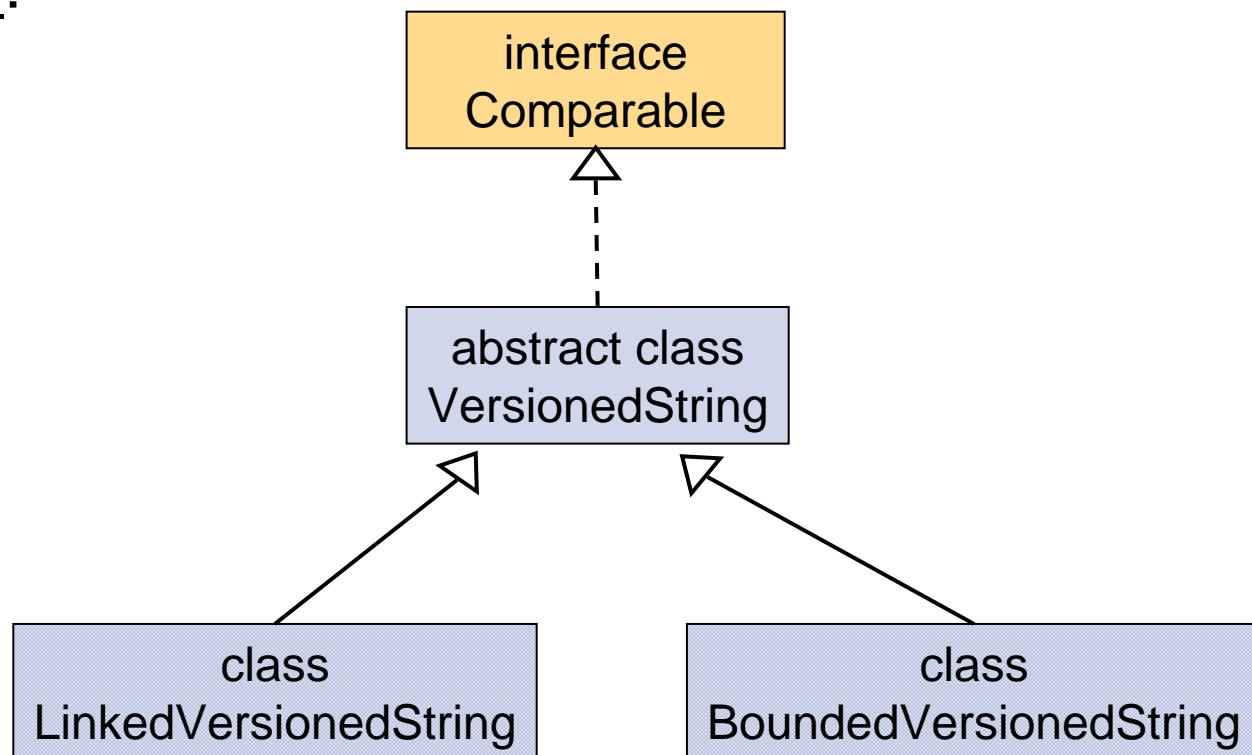
Stage 3:



# Eclipse Refactoring

## Demo (cont.)

Stage 4:



# Online Tutorials

---

- Refactoring in Eclipse

<http://www.cs.umanitoba.ca/~eclipse/13-Refactoring.pdf>

- Eclipse Handouts (on the course web-site):

[http://www.cs.tau.ac.il/courses/software1/handouts/eclipse\\_handouts.pdf](http://www.cs.tau.ac.il/courses/software1/handouts/eclipse_handouts.pdf)