Contents

1. Introduction
2. Types and Variables
3. Statements and Control Flow
4. Reading Input
5. Classes and Objects
6. Arrays
7. Methods
8. Scope and Lifetime
9. Utility classes
10. Introduction to Object-Oriented Analysis and Design
Chapter 9: Utility Classes

After this chapter you will be able to:

- Apply wrapper classes
- Import symbols from packages
- Build dynamic data structures using class Vector
Wrapper Classes

• The purpose of a wrapper class is to encapsulate (wrap up) data
• Java has the following predefined wrapper classes:
  – Integer
  – Float
  – Double
  – Character
  – Boolean
Class Integer

• Can create an **object** that encapsulates an **int** using Java’s Integer class

```java
Integer x = new Integer(4);
```

![Diagram](image_url)
Method intValue

• Can retrieve the int value by calling the intValue method

```java
Integer x = new Integer(4);
System.out.println("the int inside the object = " + x.intValue());
```

Output

the int inside the object = 4
Other Integer Methods

• public String toString();
  – returns the integer value as a string value

• public long longValue();
  – returns the integer as a long value

• public double doubleValue();
  – returns the integer as a double value
Float Class

- Can create an **object** that encapsulates a **float** using Java’s Float class

```java
Float x = new Float(4.3);
System.out.println("the float inside the object = "+x.floatValue());
```

**Output**
the float inside the object = 4.3
Packages

• In the Java API (Application Programmer Interface) classes are grouped into packages

Packages

java.lang
  - String
  - Integer
  - Boolean
  - Double
  - Float

java.net
  - URL
  - Socket

java.util
  - Vector
Package Visibility

• All classes in package `java.lang` are visible by `default`
• Classes in other packages can be made visible using the `import` statement

```
import java.util.Vector;    // import only Vector
import java.util.*;         // import all classes
// in java.util
```

• `import` is always at the top of a file
Class Vector

- A Vector object is a dynamic data structure
- Can dynamically add other kinds of objects into a Vector object
Creating Vector Objects

• Create Vector object by calling either of the constructors

```java
Vector v = new Vector();  // no initial size
```

```java
Vector v = new Vector(10);  // initial capacity for 10 objects
```
Adding Objects

• call addElement to add objects to the Vector object

```java
Vector v = new Vector();

Integer x = new Integer(4);
Integer y = new Integer(5);

v.addElement(x);
v.addElement(y);
```
Number of Objects

• method `size` returns the number of objects in the Vector

```java
import java.util.Vector;

public class VectorExample {
    public static void main (String [] args) {
        Vector v = new Vector();
        System.out.println(v.size() + " objects inside");
        v.addElement(new Integer(4));
        v.addElement(new Integer(5));
        System.out.println(v.size() + " objects inside");
    }
}
```

Output

0 objects inside
2 objects inside
Retrieving Objects

• Can index objects in Vector using `elementAt`

```
Object elementAt (int index)
```

• Objects of all classes are of type `Object`
Retrieving Objects

• Using `elementAt`

```java
Vector v = new Vector();
v.addElement(new Integer(4));
v.addElement(new Integer(5));

for (int i = 0; i < v.size(); i++) {
    Integer x = (Integer) v.elementAt(i);
    System.out.println(x.intValue());
}
```

Output

```
4
5
```
Type Casting

• Can explicitly convert the type of a value in some cases

• Syntax: \((\text{type}) \text{ value}\)

```
int a = (int) 4.3;
Vector v = new Vector();
v.addElement(new Integer(4));
Integer x = (Integer) v.elementAt(i);
System.out.println(x.intValue());
```

• Question: Why is the cast to \text{Integer} necessary?
Answer

• Because intValue is defined on objects of type Integer (not Object)

Vector v = new Vector();

v.addElement(new Integer(4));
Object x = v.elementAt(i); System.out.println(x.intValue());

• Recall: a variable’s type implies both:
  – the values that can be stored
  – the operations that can be applied
Summary

• The `Vector` class facilitates the implementation of **dynamic** data structures

• Applications
  – a shopping basket data structure for an on-line catalog
  – A rolodex
  – Any kind of list that must support future additions
It’s Exercise Time