Object-Oriented Programming
Types, Encapsulation, ArrayList

Ewan Klein

School of Informatics

Inf1 :: 2009/10

Labs

New! 'Overflow' Scheduled Lab: Wednesdays, 3.00-4.00

This Week's Labs

Lab Sessions

- Each scheduled lab will start with a 10 min demo of Eclipse.
- Eclipse can be bewildering at first …
- You are not obliged to use it, but it can be a BIG help.

Assessed Assignment

- Takes the form of this week's lab exercises.
- Released today.
- Due back noon (12.00 pm), Monday 8th Feb.
- Worth 5% of total marks.

Converting Primitive Types in Java

- Automatic type conversion: `int` $\longrightarrow$ `long`, `float` $\longrightarrow$ `double`.

Converting `int` to `long`

```
long myLongInteger;
int myInteger;
myLongInteger = myInteger;
```

Converting Primitive Types in Java

- Automatic type conversion: `int` $\longrightarrow$ `long`, `float` $\longrightarrow$ `double`.
- But cannot do automatic conversion in other direction.

Converting `int` to `long`

```
long myLongInteger;
int myInteger;
myLongInteger = myInteger;
```

Casts in Java

- Java allows us to explicitly convert in other direction (assumes we know what we are doing!)
- Use a 'type cast' of the form (T) N.

Casting `long` to `int`

```
long myLongInteger;
int myInteger;
myInteger = (int) myLongInteger;
```

NB `myLongInteger` must be in the range `Integer.MIN_VALUE` – `Integer.MAX_VALUE` (-2,147,483,647 – 2,147,483,647)

Floating Point Division, 1

- Integer division:
  - when operands of / are both `int` or `long`;
  - always gives an integer result, truncated toward 0.
  - Converting the result to a floating point number doesn't help.
- Floating Point division:
  - when operands of / are either `double` or `float`;
  - returns a fractional answer.

Division

```
5/2 // => 2
double num = 5/2;
5/2.0 // => 2.5
5.0/2 // => 2.5
```

Floating Point Division, 2

Division problem

```
int num1 = 5;
int num2 = 2;
...
double result = num1 / num2; // => 2.0
```

Floating Point Division, 2

Division problem

```
int num1 = 5;
int num2 = 2;
...
double result = num1 / num2; // => 2.0
```

Using a cast with division

```
int num1 = 5;
int num2 = 2;
...
double result = num1 / (double) num2; // => 2.5
```

Enhanced for loop, 1

```
int[] numbers = {2, 5, 6, 1, 0, 5};
```

Ordinary for loop

```
for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
```

Enhanced for loop

```
for (int num : numbers) { //num is the loop variable
    System.out.println(num);
}
```

Enhanced for loop, 1

Also called **foreach** loop; easier to use with arrays and other 'iterables'.
General form:

`for (` *variable declaration* `:` *iterable* `) { ... }`

NB the variable must have same type as elements in *iterable*.

- `:` often read as 'in'
- On each iteration, an element of the iterable gets assigned to the loop variable.
- Loop gets executed once for each element in the iterable.
- Don't need to initialize loop variable or set termination condition.
- But you can't keep track (via the loop counter) of where you are in the iteration (e.g., first, last, etc)

Enhanced for loop, 3

Another Example: Right

```
String[] words = {”hello”, ”world”, ”we”, ”can”, ”do”, ”it”};
for (String w : words) {
    System.out.println(w);
}
```

Another Example: Wrong

```
String[] words = {”hello”, ”world”, ”we”, ”can”, ”do”, ”it” };
for (int w : words) {
    System.out.println(w);
}
```

Information Hiding, 1

`public` vs. `private`

Exposing an Instance Variable

```
public class CashCard {
    int balance = 150; // public by default
}
```

Information Hiding, 2

Accessing an Instance Variable

```
public class Fraudster {
    int gotIt;
    public void getMoney(CashCard cc) {
    gotIt += cc.balance;
    cc.balance -= cc.balance;
    }
    public static void main(String[] args) {
        CashCard yrCard = new CashCard();//yrCard balance is 150
        Fraudster fraud = new Fraudster();
        fraud.getMoney(yrCard);//yrCard balance is 0!
    }
}
```

Access Control

- All the instance variables and methods (i.e., members) of a class can be used within the body of the class.
- Access control modifiers: specify access control rules.

`public`:	member is accessible whenever the class is accessible.
`private`:	member is only accessible within the class.
default:	amounts to `public` for current purposes.

Information Hiding, 3

Hiding an Instance Variable

```
public class SecureCashCard {
    private int balance;
}
```

Frustrated Fraudster

```
public class Fraudster {
    int gotIt;
    public void getMoney(SecureCashCard cc) {
    gotIt += cc.balance; //Forbidden
    cc.balance -= cc.balance; //Forbidden
    }
...
}
... The field SecureCashCard.balance is not visible
```

Getters and Setters

Adding a getter

```
public class FairlySecureCashCard {
    private int balance;
    public int getBalance() {//Allow others to peek
        return balance;
    }
}
```

Getters and Setters

Adding a getter

```
public class NotSecureCashCard {
    private int balance;
    public int getBalance() {//Allow others to peek
        return balance;
    }
    public void setBalance(int balance) {//Allow others to modify!
        this.balance = balance;
    }
}
```

Encapsulation

- loose coupling
- protected variation
- Exporting an API — Application Programming Interface
  - the classes, members etc by which some program is accessed
  - any client program can use the API
  - the author is committed to supporting the API

Squeezing into an array

- Length of array is fixed at creation time.
- Can't be expanded.
- Can't be shrunk.
- Array is part of Java language — uses special syntax.
- E.g., `myArray[i]` for accessing the ith element.

Squeezing into an array

ArrayList, o

Adding Elements

```
ArrayList<String> cheers = new ArrayList<String>();
```

`<String>` is a type parameter.

More generally:

- `ArrayList<E>` is a class.
- Stuff in angle brackets tells us the type of element.
- `ArrayList` has various methods, which allow us to:
  - keep on adding new elements;
  - delete elements.

ArrayList, 1

Adding Elements

```
ArrayList<String> cheers = new ArrayList<String>();
cheers.add("hip");
cheers.add("hip");
cheers.add("hooray");
int n1 = cheers.size(); // n1 is set to 3
```

Append each element to the end of the list.

Elements of `cheers`: `"hip"`, `"hip"`, `"hooray"`

ArrayList, 2

Index of first occurrence

```
int n2 = cheers.indexOf("hip");  // n2 is set to 0
```

Adding Element at an Index

```
cheers.add(1, "hop"); // 2nd "hip" gets shunted along
```

Elements of `cheers`: `"hip"`, `"hop"`, `"hip"`, `"hooray"`

ArrayList, 3

Contains

```
boolean isHip = cheers.contains("hip");  // isHip is true
```

ArrayList, 4

Remove

```
cheers.remove("hip");  // removes first occurrence of "hip"
```

Elements of `cheers`: "hop", "hip", "hooray"

ArrayList, 5

Enhanced for again

```
for (String s : cheers) {
     System.out.print(s + "\thas index: ");
     System.out.println(cheers.indexOf(s));
}
```

Output

```
hop     has index: 0
hip     has index: 1
hooray  has index: 2
```

Full vs. Simple Names

Fully Qualified Name

```
java.util.ArrayList<String> cheers =
       new java.util.ArrayList<String>();
```

Simple Name

```
ArrayList<String> cheers = new ArrayList<String>();
```

Import

To be able to use the simple name, add an import statement at the top of your file.

Import example

```
import java.util.ArrayList;
```

Import example — Wrong!

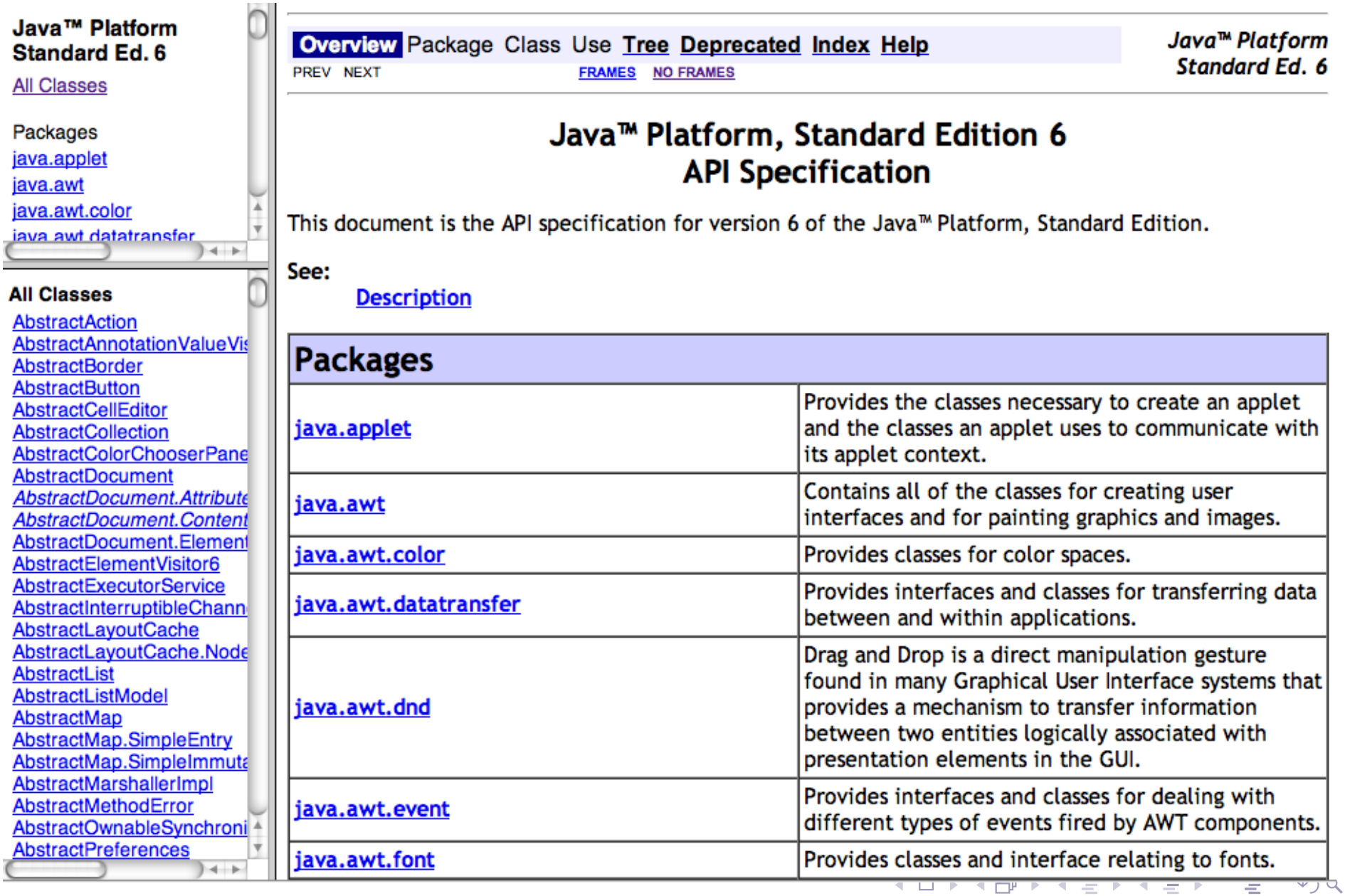
```
import java.util.ArrayList<String>; // Don't use parameter
```

Java API

Look at sample Javadoc web page.

`http://java.sun.com/javase/6/docs/api/`

Java API: Top Node



Java™ Platform Standard Ed. 6

All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer

All Classes
AbstractAction
AbstractAnnotationValueVis
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPane
AbstractDocument
AbstractDocument.Attribute
AbstractDocument.Content
AbstractDocument.Element
AbstractElementVisitor6
AbstractExecutorService
AbstractInterruptibleChann
AbstractLayoutCache
AbstractLayoutCache.Node
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmuta
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchroni
AbstractPreferences

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Java™ Platform Standard Ed. 6

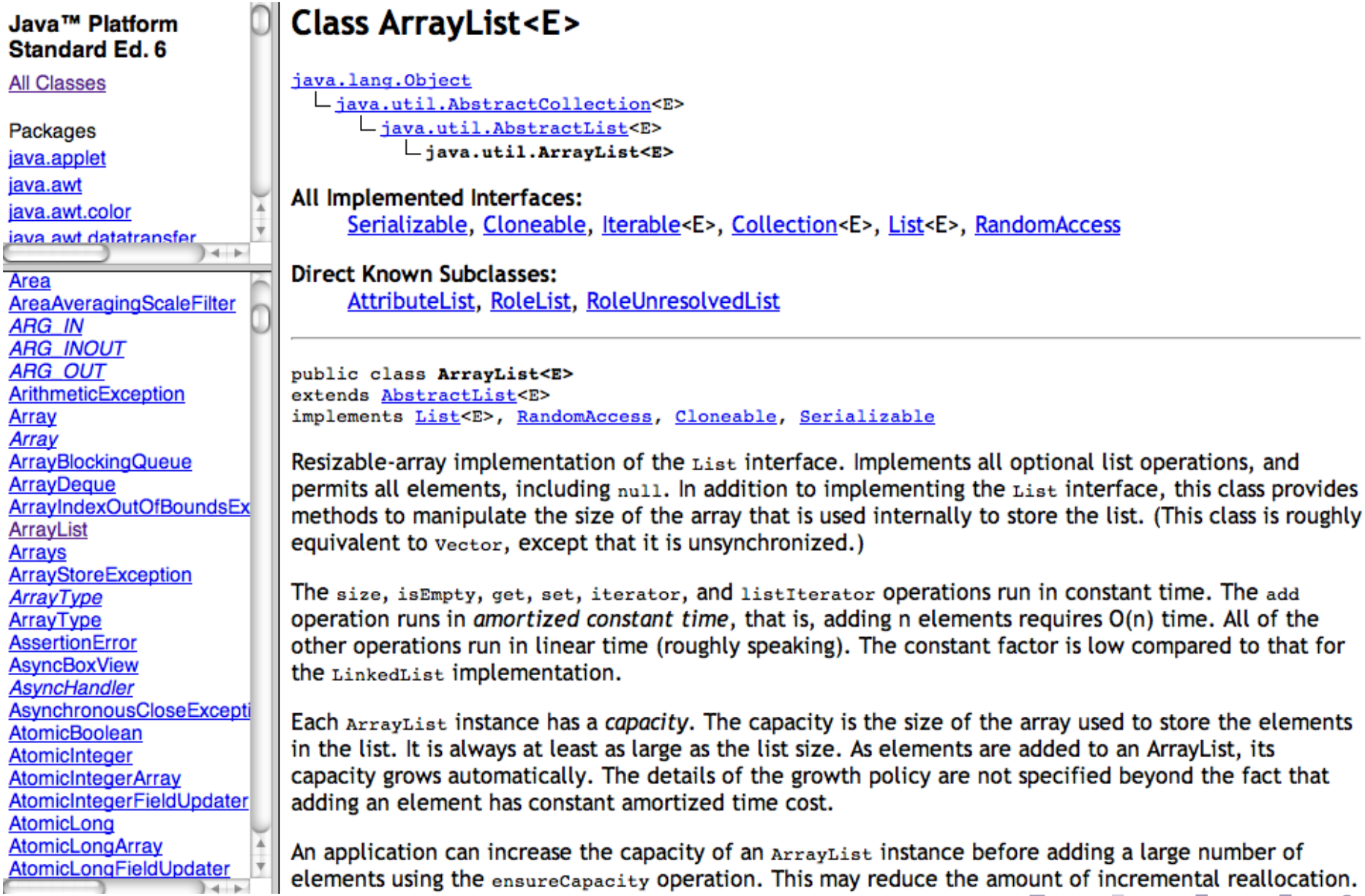
Java™ Platform, Standard Edition 6 API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:
Description

Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.

Java API: Entry for ArrayList



Reading

- encapsulation: Chapter 4 HFJ
- casts: Chapter 5 HFJ
- enhanced `for`: Chapter 5 HFJ
- `ArrayList`: Chapter 6 HFJ
- Java API: Chapter 6 HFJ