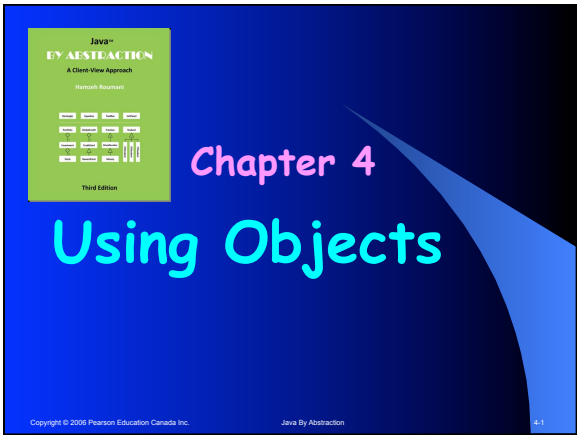
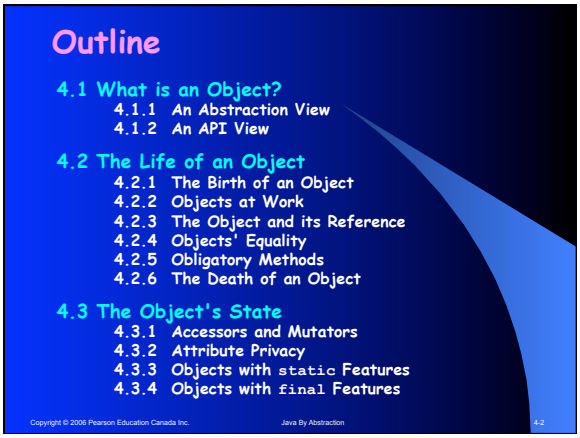
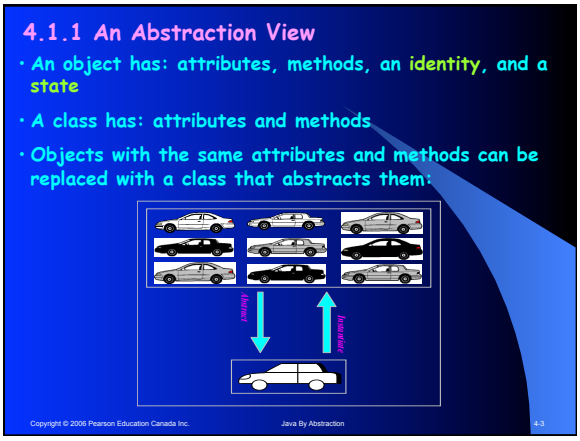

Java
BY ABSTRACTION
A Client-View Approach
Hamish Roumani
Third Edition
Chapter 4
Using Objects
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-1


Outline
4.1 What is an Object?
4.1.1 An Abstraction View
4.1.2 An API View
4.2 The Life of an Object
4.2.1 The Birth of an Object
4.2.2 Objects at Work
4.2.3 The Object and its Reference
4.2.4 Objects' Equality
4.2.5 Obligatory Methods
4.2.6 The Death of an Object
4.3 The Object's State
4.3.1 Accessors and Mutators
4.3.2 Attribute Privacy
4.3.3 Objects with static Features
4.3.4 Objects with final Features
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-2


4.1.1 An Abstraction View
• An object has: attributes, methods, an identity, and a state
• A class has: attributes and methods
• Objects with the same attributes and methods can be replaced with a class that abstracts them:
Abstract
Instantiate
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-3

4.1.2 An API View

- The API of an instantiable class has three sections:
  - A Constructor Section
  - A Field Section
  - A Method Section
- Constructors allow us to instantiate the class and get an object; i.e. add identity and state
- A constructor section *looks like* a method but:
  - There is no return column (not even `void`)
  - Constructor name = Class name

 Java By Abstraction 4-4

4.2.1 The Birth of an Object

A four-step process:

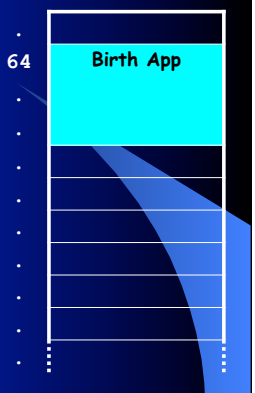
1. Locate the Class
   ```
   import type.lib.Fraction;
   ```
2. Declare a Reference
   ```
   Fraction f;
   ```
3. Instantiate the Class
   ```
   new Fraction(3, 5)
   ```
4. Assign the Reference
   ```
   f = new Fraction(3, 5);
   ```

 Java By Abstraction 4-5

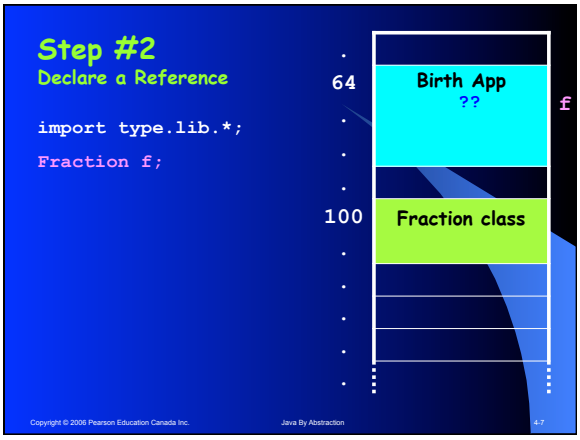
Step #1

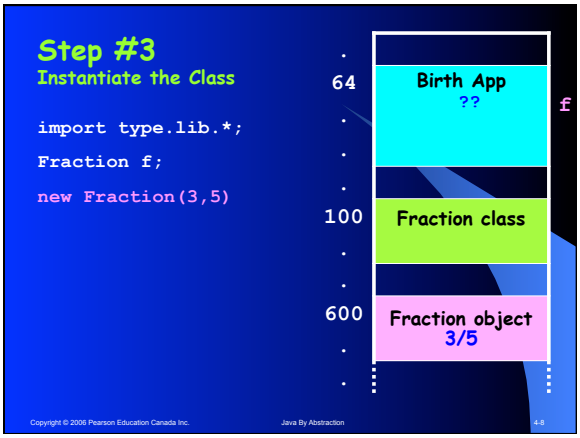
Locate the Class

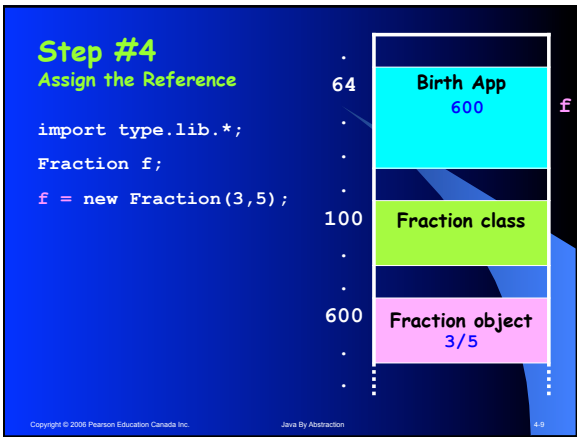
```
import type.lib.*;
```



 Java By Abstraction 4-6

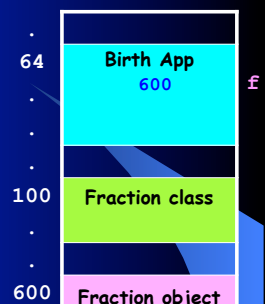

Step #2
Declare a Reference
import type.lib.*;
Fraction f;
64
100
Birth App
??
Fraction class
f
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-7


Step #3
Instantiate the Class
import type.lib.*;
Fraction f;
new Fraction(3,5)
64
100
600
Birth App
??
Fraction class
Fraction object
3/5
f
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-8


Step #4
Assign the Reference
import type.lib.*;
Fraction f;
f = new Fraction(3,5);
64
100
600
Birth App
600
Fraction class
Fraction object
3/5
f
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-9

Step #4
Assign the Reference

```
import type.lib.*;
Fraction f;
f = new Fraction(3,5);
```



A reference is a pointer

 Java By Abstraction 4-10

4.2.2 Objects at Work

- Accessing Field
  `reference.field`
- Invoking Methods
  `reference.method(…)`

Unlike static/utility classes, we access and invoke on the reference, not on the class.

 Java By Abstraction 4-11

Examples

- Create 8/6 and invoke methods
- Note the role of `separator` and `isQuoted`
- Compute:

$$\frac{\frac{5}{3}\times\frac{7}{6}}{\frac{31}{45}}+\frac{3}{4}$$

 Java By Abstraction 4-12

4.2.3 The Object Reference

Variables of primitive types hold values:

```
int x = 5;
int y = x;
x = 10;
// at this stage y remains 5
```

Variables of non-primitive types (references) hold addresses of objects, not the objects themselves.

 Java By Abstraction 4-13

Aliases

Many variables can point at the same object:

```
Fraction f1;
f1 = new Fraction(3, 5);
Fraction f2;
f2 = f1;
```

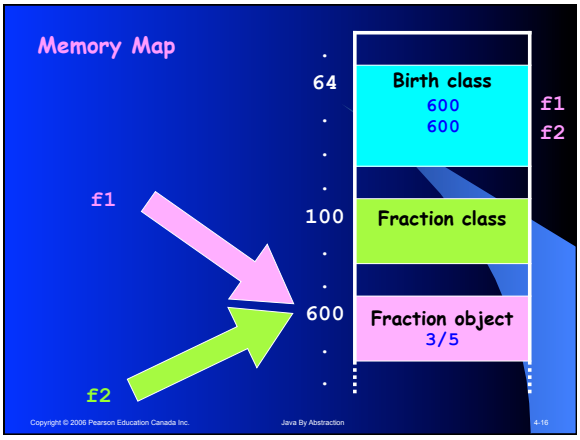
If the object is changed through `f1`, the change will be seen by `f2`.

 Java By Abstraction 4-14

Example

```
Fraction f1;
f1 = new Fraction(3, 5);
Fraction f2;
f2 = f1;
f1.separator = "|";
System.out.println(f2.toString());
```

 Java By Abstraction 4-15



Null References and Orphans

```
Fraction f1;
f1 = new Fraction(3, 5);
Fraction f2;
f2 = f1;
f1 = null;
System.out.println(f1.toString());
System.out.println(f2.toString());
f2 = null;
```

Note that `null` is a literal (just like `true` and `false`) whose type is compatible with any non-primitive type.

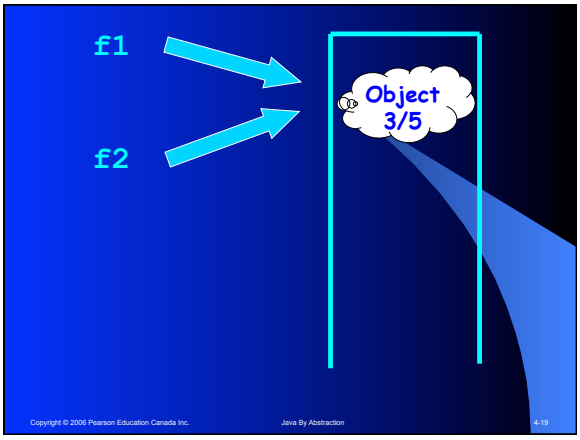
 Java By Abstraction 4-17

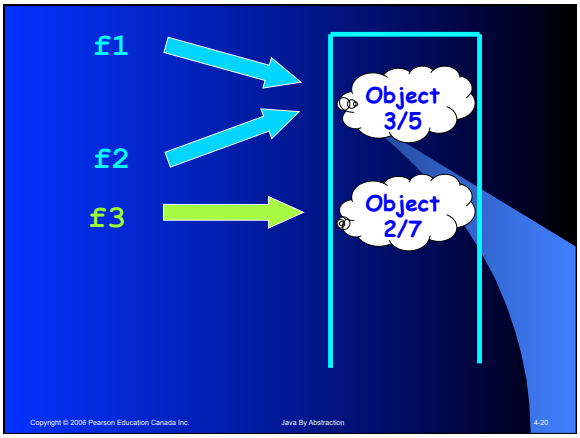
4.2.4 Object Equality

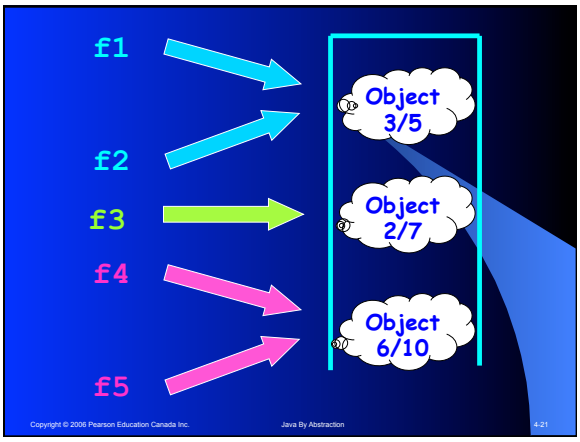
The == operator determines whether two object references are pointing at the same memory block:

```
Fraction f1 = new Fraction(3, 5);
Fraction f2 = f1;
Fraction f3 = new Fraction(2, 7);
Fraction f4 = new Fraction(6, 10);
Fraction f5 = f4;
System.out.println(f1 == f2);
System.out.println(f4 == f5);
System.out.println(f4 == f1);
```

 Java By Abstraction 4-18







== versus equals

```
Fraction f1 = new Fraction(3, 5);
Fraction f2 = f1;
Fraction f3 = new Fraction(2, 7);
Fraction f4 = new Fraction(6, 10);
Fraction f5 = f4;
System.out.println(f1 == f2);
System.out.println(f4 == f5);
System.out.println(f4 == f1);

System.out.println(f4.equals(f1));
```

The `equals` method determines whether two objects are equal in the eyes of their class.

 Java By Abstraction 4-22

Java provides a default equals method for classes that do not have one of their own. This "default" equals method behaves the same as ==:

```
FractionNS x = new FractionNS(4, 5);
FractionNS y = new FractionNS(4, 5);

boolean equalRef = (x == y);
boolean equalObj = x.equals(y);
```

 Java By Abstraction 4-23

4.2.5 Obligatory Methods

Certain methods are available in all classes, either directly (provided by the class itself) or indirectly (provided by Java). Two such methods are:

toString
- -Default behaviour: same as ==
- -Auto-invoked by output methods

equals
- -Default behaviour: class name and the object's memory address in hex.

 Java By Abstraction 4-24

4.2.6 The Death of an Object

Destroy the object-reference connection by:

- Exiting the scope of the reference
- Setting the reference to null
- Pointing the reference elsewhere

```
Fraction x = new Fraction(3, 5);
Fraction y = x;
Fraction z = x;
{
   Fraction t = x;
}
y = null;
z = new Fraction(4, 7);
```

 Java By Abstraction 4-25

We can destroy the object itself (indirectly) by orphaning it:

```
Fraction x = new Fraction(3, 5);
Fraction y = x;
Fraction y = new Fraction(4, 7);
x = null;
```

 Java By Abstraction 4-26

4.3.1 Accessors and Mutators

Key points to remember:

- A class has attributes and methods
- The object's state is held in the attributes
- Implementers make all non-final attributes private and provide accessors and mutators to enable clients to access the state.
- Accessors provide read-only access
- Mutators allow clients to mutate the state

See the `type.lib.Item` class

 Java By Abstraction 4-27

4.3.2 Objects with `static` features

Some features (attributes and/or methods) in a class can be `static`. Such features:

- stay in the class
- Are shared by all instances
- Should be invoked on the class, *not* on the object reference (even though the compiler tolerates the latter).

See `isQuoted` in `type.lib.Fraction`.

 Java By Abstraction 4-28

```
Fraction f = new Fraction(3, 2);
System.out.println(f.toProperString());
f.isQuoted = false;
System.out.println(f.toProperString());
```

The output:

```
"1 1/2"
1 1/2
```

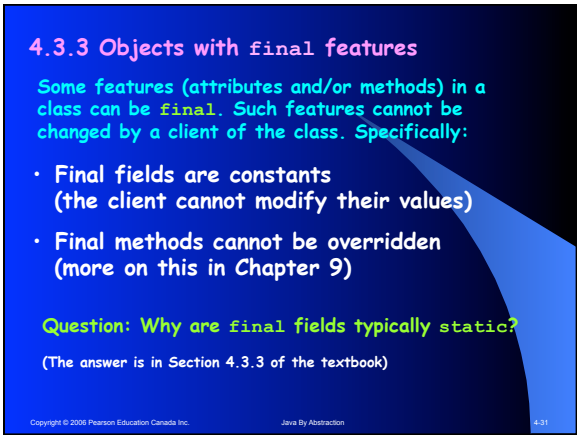
 Java By Abstraction 4-29

```
Fraction f = new Fraction(3, 2);
f.isQuoted = true;

Fraction g = new Fraction(5, 2);
g.isQuoted = false;

System.out.println(f.toProperString());
System.out.println(g.toProperString());
```

The output:

```
1 1/2
2 1/2
```

 Java By Abstraction 4-30


4.3.3 Objects with final features
Some features (attributes and/or methods) in a class can be final. Such features cannot be changed by a client of the class. Specifically:
• Final fields are constants (the client cannot modify their values)
• Final methods cannot be overridden (more on this in Chapter 9)
Question: Why are final fields typically static?
(The answer is in Section 4.3.3 of the textbook)
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
4-31