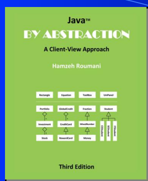


Chapter 3
Using APIs

Outline

3.1 Anatomy of an API
- 3.1.1 Overall Layout
- 3.1.2 Fields
- 3.1.3 Methods

3.2 A Development Walkthrough
- 3.2.1 The Development Process
- 3.2.2 The Mortgage Application
- 3.2.3 Output Formatting
- 3.2.4 Relational Operators
- 3.2.5 Input Validation
- 3.2.6 Assertions

3.3 General Characteristics of Utility Classes
- 3.3.1 Memory Diagram
- 3.3.2 Advantages of Utility Classes
- 3.3.3 Case Study: Dialog I/O

3.1.1 Overall Layout

Packages	Details
	The Class section
Classes	The Field section
	The Constructor section
	The Method section

3.1.2 Fields

Field Summary

static double	**PI** The `double` value that is closer than any other to *pi,* the ratio of the circumference of a circle to its diameter.

Field Detail

PI

`public static final double PI`

The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.

See Also: Constant Field Values

 Java By Abstraction 3-4

3.1.3 Methods

The Math class of java.lang

Method Summary

static double	**abs**(double a) Returns the absolute value of a `double` value.
static int	**abs**(int a) Returns the absolute value of an `int` value.
static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.

The Scanner class of java.util

Method Summary

double	**nextDouble**() Scans the next token of the input as an `double`.
int	**nextInt**() Scans the next token of the input as an `int`.
String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.
long	**nextLong**() Scans the next token of the input as an `long`.

 Java By Abstraction 3-5

Method Detail

abs

`public static double abs(double a)`

Returns the absolute value of a `double` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is `NaN`, the result is `NaN`.

Parameters:
a - the argument whose absolute value is to be determined

Returns:
the absolute value of the argument.

 Java By Abstraction 3-6

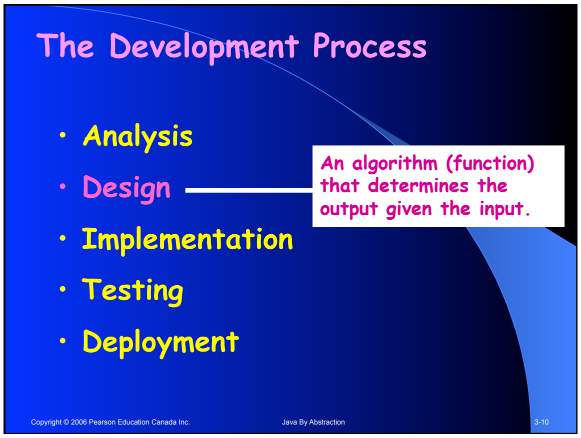
Key points to remember about methods

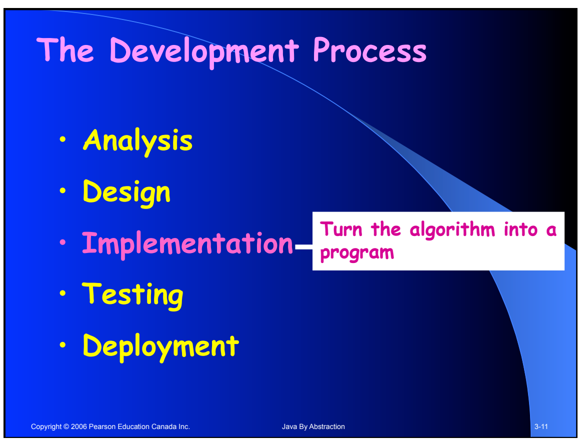
- **Parameters are Passed by Value**
  Values stored in your variables cannot be inadvertently changed by passing the variables to a method
- **Methods can be Overloaded**
  A class cannot have two methods with the same signature (even if the return is different). Hence, can have two methods with the same name (but different parameters)
- **Binding with Most Specific**
  To bind `C.m(…)` the compiler locates C (or else issues No Class Definition Found) and then locates m(…) in C (or else issues Cannot Resolve Symbol). If more than one such m is found, it binds with the "most specific" one.

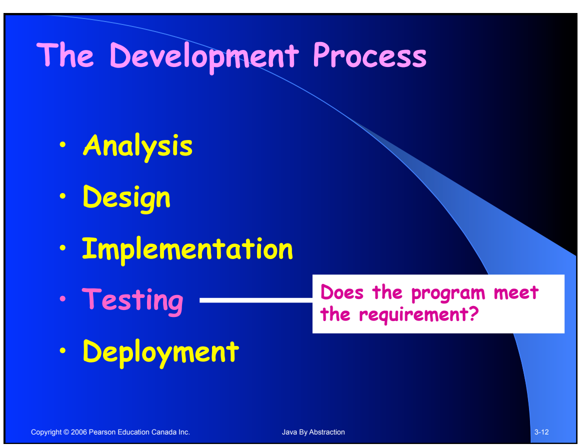
3.2 A Development Walkthrough

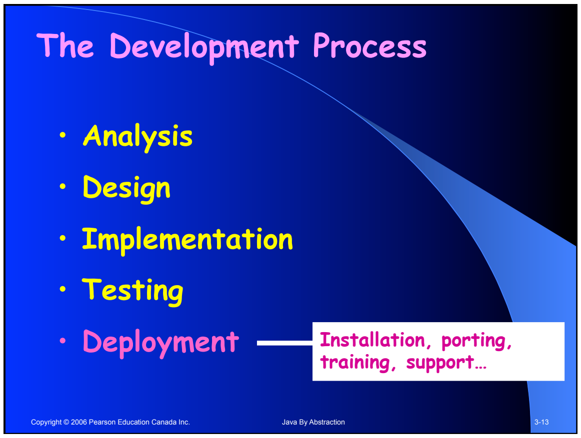
3.2.1 The Development Process

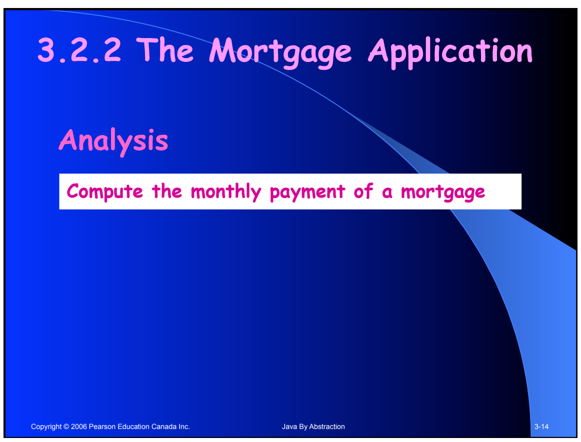
- Analysis — The Requirement: Input & its validation Output & its formatting
- Design
- Implementation
- Testing
- Deployment

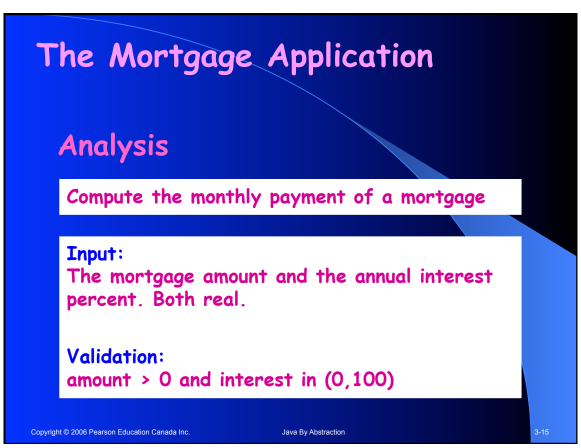

The Development Process
• Analysis
• Design
An algorithm (function) that determines the output given the input.
• Implementation
• Testing
• Deployment
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-10


The Development Process
• Analysis
• Design
• Implementation
Turn the algorithm into a program
• Testing
• Deployment
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-11


The Development Process
• Analysis
• Design
• Implementation
• Testing
Does the program meet the requirement?
• Deployment
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-12


The Development Process
• Analysis
• Design
• Implementation
• Testing
• Deployment
Installation, porting, training, support…
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-13


3.2.2 The Mortgage Application
Analysis
Compute the monthly payment of a mortgage
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-14


The Mortgage Application
Analysis
Compute the monthly payment of a mortgage
Input:
The mortgage amount and the annual interest percent. Both real.
Validation:
amount > 0 and interest in (0,100)
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
3-15

The Mortgage Application

Analysis

Output:
The monthly payment.

Formatting:
rounded to the nearest cent and displayed with a thousands separator

 Java By Abstraction 3-16

The Mortgage Application

Analysis

Sample run of the proposed system:

```
Enter the amount ... 285000
The annual interest percent ... 3.75
The monthly payment is: $1,465.27
```

 Java By Abstraction 3-17

Design

$$P = \frac{rA}{1 - \dfrac{1}{(1+r)^n}}$$

P is the monthly payment, **r** is the monthly interest rate, **A** is the mortgage amount, and **n** is the number of months.

 Java By Abstraction 3-18

Implementation

We delegate as follows:

- A class to take care of prompts and inputs
- Ignore validation for now
- We'll do the computation ourselves with the help of a class that computes powers
- A class for output
- Ignore formatting for now

Implementation Notes

- The importance of prompting
- Using print versus println
- The next methods
- Converting from an annual percent to a monthly rate
- Why hard-coded constants like 12 are a source of confusion; using final.

3.2.3 Output Formatting

The `printf` method

- The first parameter holds format specifiers
- Each specifier has the form: **%[flags][width][.precision]conversion**
- The conversion letter can be d,f,s, or n
- The flag can be , or 0
- The width specifies the field width and the precision specifies the number of decimals

Example: `output.printf("%,6.2f", x)`

3.2.4 Relational Operators

< <= > >= Numeric operands

== != Operands of any type

All relational operands are "odd" in that they violate closure: no matter what the operand type is, the result type is always **boolean**.

Operator Precedence

Precedence	Operator	Operands	Syntax	**true** if
-7 →	<	numeric	x < y	x is less than y
	<=	numeric	x <= y	x is less than or equal to y
	>	numeric	x > y	x is greater than y
	>=	numeric	x >= y	x is greater than or equal to y
	instanceof	x instanceof C is true if object reference x points at an instance of class C or a subclass of C.		
-8 →	==	any type	x == y	x is equal to y
	!=	any type	x != y	x is not equal to y

3.2.5 Input Validation

Invalid inputs are the cause of most errors in programs. Therefore, upon encountering one, a program must either:

- Print a message and **end**
- Print a message then allow the user to **retry** several times or decide to abort.
- Trigger a runtime **error**; i.e. crash.

For now, let us use the 3rd via a method in Toolbox:

```
static void crash(boolean, String)
```


3.2.6 Assertions

A simple yet powerful tool to guard against errors that arise from misunderstandings.

Whenever you believe that some non-trivial condition is true, assert it, e.g.

```
assert payment >=0;
```

You cannot assert a validation because user input is not under your control. Hence, do not confuse assert (a Java statement) with crash (a method).

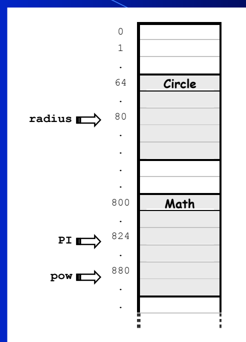
3.3 General Characteristics of Utility Classes

3.3.1 Memory Diagrams

Let us compile and load the program, Circle, which uses a field and a method in the Math utility class.

```
import java.util.Scanner;
import java.io.PrintStream;
public class Circle
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);
      PrintStream output = System.out;
      output.print("Enter radius: ");
      int radius = input.nextInt();
      output.println(Math.PI * Math.pow(radius, 2));
   }
}
```


Memory Diagrams



3.3.2 Advantages of Utility Classes

Simplicity

- To access a static field f in a class C, write: C.f
- To invoke a static method m in a class C, write C.m(…)
- There is only one copy of a static class in memory

Suitability

- A utility class is best suited to hold a groups of methods that do not hold state, e.g. java.lang.Math.
- Even in non-utility classes, static is best suited for features that are common to all instances, e.g. the MAX_VALUE field and the parseInt method of the (non-utility) class: Integer.

3.3.3 Case Study: Dialog I/O

Two static methods in:

`javax.swing.JOptionPane`

- To display a message:
  `void showMessage(null, message)`
- To prompt for and read an input:
  `String showInputDialog(null, prompt)`

Note that showInputDialog returns a String. Hence, if you use it to read a number, you must invoke one of the parse methods in the corresponding wrapper class.