Java By Abstraction: Chapter 3

Using APIs
(Application Programming Interfaces)

Some examples and/or figures were borrowed (with permission)
from slides prepared by Prof. H. Roumani

API Layout

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

API – 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

API – Methods

Math class in java.lang package

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.

Scanner class in java.util package

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.

Passing Parameters

- Syntax
  - In API: *methodName*(*paramType param*)
  - In Code: *methodName*(*identifierOrLiteral*)
- Example
  - In API: print(String s)
  - In Code:

    String **prompt** = “Enter value: ”;
    System.out.print(**prompt**); // OR
    System.out.print(“**Enter value:** ”);

Passing Parameters

- Similar to assignment statement
  - Parameter type evaluated
  - Promoted (if necessary)
  - Resulting value passed to method
- Technique referred to as **pass-by-value**

Overloading Methods

- Method signature
  - Method name + parameter types
  - Must be unique within a class regardless of return type
- Overloaded methods
  - Same name, but take different parameters
- Example
  - Math.abs(double a)
  - Math.abs(long a)

Binding with Most Specific Method

- Example: C.m(…)
  - Compiler locates class C
    - Possible "cannot find symbol" error if C not found
  - Compiler locates m(…)
    - Possible "cannot find symbol" error if m(…) not found
  - If more than one m(…), compiler binds with the "most specific" one
    - E.g., the one that does not need casting

Input Validation

- Upon encountering invalid input, you could
  - Terminate the program and display a message
  - Explain the problem and have user re-enter input
  - Trigger an appropriate runtime error – crash
- For simplicity's sake, let's simply crash
  - type.lib.ToolBox.crash(boolean b, String s)
- Example
  - ToolBox.crash(amount < 0, "A negative amount!");

Parsing Strings as Numbers

- Sometimes, numerical values are provided as Strings
- These Strings can be interpreted as primitives

- Wrapper classes
  - Encapsulate primitives as objects (sometimes useful)
  - Provide static methods to parse values
  - E.g., Integer.parseInt(…), Double.parseDouble(…)

Dialog Input/Output

- Programs often have graphical user interface elements (e.g., windows, menus, dialogs)
- GUI design in EECS1030 and EECS3461

- Simple GUI I/O using JOptionPane dialogs
  - Output: JOptionPane.showMessage(…)
  - Input: JOptionPane.showInputDialog(…)
    - Returns input as a String
    - Numeric values must be parsed

ChangeMaker2 (with Dialog I/O)

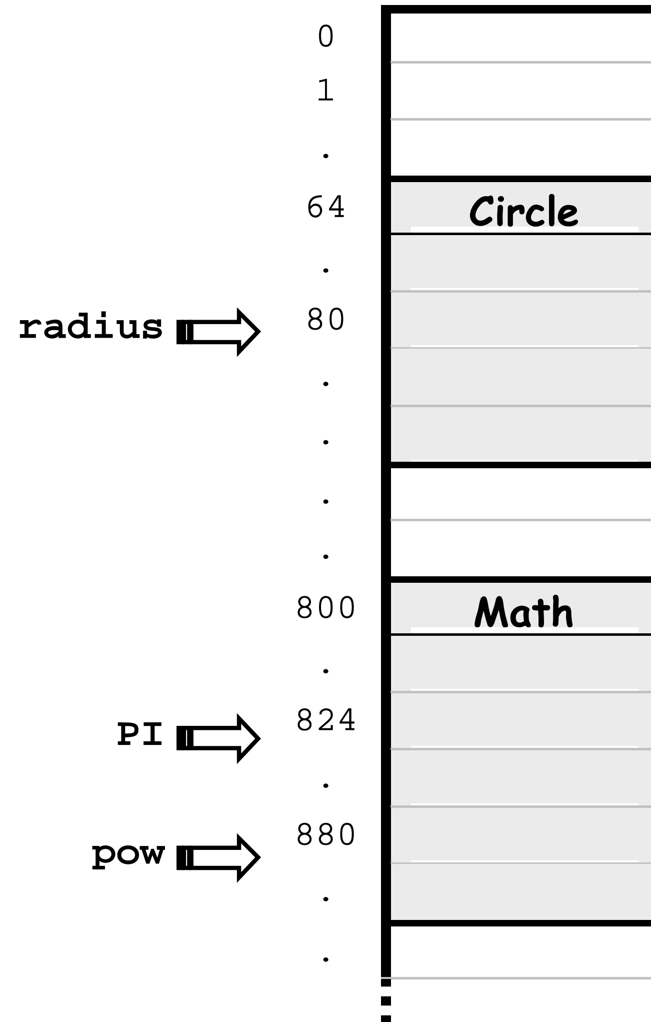
- Modify ChangeMaker.java to use dialog I/O
- (To be demonstrated in class)

Memory Diagrams

- The Circle class (below) uses an int 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 (2)



Advantages of a Utility Class

- 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
  - Best to hold methods that do not hold state
    - E.g., java.lang.Math
  - Best for features common to all instances
    - E.g., MAX_VALUE field and parseInt method in Integer

Development Process

- Task Analysis
- Algorithm Design
- Code Implementation
- Program Testing and Debugging

Task Analysis

- Input
  - amount: the present value
  - rate: the interest rate (% per annum)
- Output
  - The monthly payment formatted to two decimal places with a thousands separator
- Throws
  - Exception if amount < 0
  - Exception if rate <= 0
  - Exception if rate >= 100

Algorithm Design

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

Code Implementation

- (To be demonstrated in class)

Program Testing and Debugging

- Pick example inputs for the program
- Use a calculator to determine expected results
- Compare expected results with actual ones
- Identify any sources of error in the program
  - Debug using print statements to output variable values
- Correct any errors
- Retest program