The Java API & Data Representation

CSC 1051 – Data Structures and Algorithms I
Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Course website:
http://www.csc.villanova.edu/~map/1051/

Packages

- For purposes of accessing them, classes in the Java API are organized into packages
- These often overlap with specific APIs
- Examples:

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support, String and Math classes</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>java.text</td>
<td>Text utilities (eg formatting)</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>javafx.scene.shape</td>
<td>Graphical shapes</td>
</tr>
<tr>
<td>javafx.scene.control</td>
<td>GUI controls</td>
</tr>
</tbody>
</table>

The Java class library or Java API (Application Programming Interface)

The Math Class

- The Math class is part of the java.lang package and contains methods for mathematical functions
  - No need to import anything!
  - The Math class methods are static
  - Static methods are invoked through the class name

  ```java
  value = Math.cos(phi) + Math.sqrt(delta);
  ```

See Quadratic.java
Some methods from the Math class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(double a)</code></td>
<td>Absolute value of <code>a</code></td>
</tr>
<tr>
<td><code>max(double a, double b)</code></td>
<td>Maximum of <code>a</code> and <code>b</code></td>
</tr>
<tr>
<td><code>min(double a, double b)</code></td>
<td>Minimum of <code>a</code> and <code>b</code></td>
</tr>
<tr>
<td><code>sin(double theta)</code></td>
<td>Sine function</td>
</tr>
<tr>
<td><code>cos(double theta)</code></td>
<td>Cosine function</td>
</tr>
<tr>
<td><code>tan(double theta)</code></td>
<td>Tangent function</td>
</tr>
<tr>
<td><code>exp(double a)</code></td>
<td>Exponential function</td>
</tr>
<tr>
<td><code>log(double a)</code></td>
<td>Natural log (base e)</td>
</tr>
<tr>
<td><code>sqrt(double a)</code></td>
<td>Square root of <code>a</code></td>
</tr>
<tr>
<td><code>E</code></td>
<td>Value of e (constant)</td>
</tr>
<tr>
<td><code>PI</code></td>
<td>Value of π (constant)</td>
</tr>
</tbody>
</table>

Also defined for `int`, `float`, and `long`.

The Strings Class

- Strings are objects defined by the `String` class
  - "This is a string literal."
  - "123 Main Street"
  - "X"

- The `String` class has many methods that can be used to process text. Examples:
  - Finding the length of a string
  - Finding the char at a certain position of a string
  - Producing an all-caps version of a string

Invoking String Methods

- As with other kinds of objects, we use the dot operator to invoke a String's methods:

  ```java
  String name = "Betsy";
  int numOfCharsInName = name.length();
  ```

More String Methods

```java
String name = "Betsy";
char initial = name.charAt(0);
String newName = name.replace('s', 't');
String capsName = name.toUpperCase();
int comp = name.compareTo(newName);
```
Example: Palindrome tester

- Problem: Input a string, determine whether it is a palindrome, i.e.:
  - first char is the same as last char
  - 2nd char is the same as 2nd last char
  - and so on...

- How to express this as an algorithm?
- How to implement it?

```java
import java.util.Scanner;

public class PalindromeTester {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a potential palindrome:");
        String str = scan.nextLine();
        int left = 0;
        int right = str.length() - 1;
        while (str.charAt(left) == str.charAt(right) && left < right) {
            left++;
            right--;
        }
        if (left < right) {
            System.out.println("NOT a palindrome");
        } else {
            System.out.println("palindrome");
        }
    }
}
```

Sample Run

Enter a potential palindrome: radar
palindrome
Enter another palindrome (y/n)? y
Enter a potential palindrome: able was I ere I saw elba
palindrome.
Test another palindrome (y/n)? y
Enter a potential palindrome: abracadabra
NOT a palindrome.
Test another palindrome (y/n)? n

The import Declaration

- When you want to use a class from a package, you could use its fully qualified name
  ```java
  java.util.Scanner
  ```

- Or you can import the class, and then use just the class name
  ```java
  import java.util.Scanner;
  ```

- To import all classes in a particular package, you can use the * wildcard character
  ```java
  import java.util.*;
  ```

The Random Class

- Part of the `java.util` package, so import it
  ```java
  import java.util.Random;
  ```

- Create a Random object named `gen`:
  ```java
  Random gen = new Random();
  ```

- Use `Random` method `nextInt()` to generate a random number:
  ```java
  int a = gen.nextInt(4);
  // integer in range [0,1,2,3]
  ```
What is a random number?

“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”
- John Von Neumann

“God does not play dice.”
- Albert Einstein

The Random class provides methods that generate pseudorandom numbers

Example: counting “snake eyes”

// Roll two dice 100,000 times and count how many // times you roll snake eyes, i.e., two 1's.
Random gen = new Random();
int trial = 0, count = 0;
while (trial < 100000)
{
    int die1 = gen.nextInt(6) + 1;
    int die2 = gen.nextInt(6) + 1;
    if (die1 == 1 && die2 == 1)
        count++;
    trial++;
}
System.out.println("Probability of snake eyes = " + (double)count/100000);

Summary: Generating pseudorandom numbers

Random gen = new Random();
int a = gen.nextInt(4);
    // integer in range [0,1,2,3]
int b = gen.nextInt(4) + 1;
    //int in range [1,2,3,4]
int c = gen.nextInt();
    // int in range [-2147483648 ... 2147483647]
float d = gen.nextFloat();
    //float in range [0,1), eg: 0.4589
double e = Math.random();
    //double in range [0,1), eg: 0.4589
int f = (int) (Math.random() * 4);
    // integer in range [0,1,2,3] (same as a, above)

See also RandomNumbers.java

Monte Carlo simulation example: approximate the value of π

final long MAXPOINTS = 100000000;   // number of random points
long count = 0;
long inCircleCount = 0;       // counts points inside circle
double x, y;   // points in interval (0,1)
Random toss = new Random();
while (count < MAXPOINTS) {
    x = toss.nextDouble();  // toss in quadrant
    y = toss.nextDouble();
    if ((x*x + y*y) < 1) // inside unit circle
        inCircleCount++;
    count++;
}
double myPI = 4.0 * inCircleCount / MAXPOINTS;
System.out.println("Value of pi = " + myPI);
System.out.println("Math.PI = " + Math.PI + ");
Declaring Variables, revisited

- Examples of variable declarations:
  ```java
  int count = 0;
double mpg;
String title;
Graphics page;
Color aquamarine;
Scanner scan;
  ```

- A **class name** can be used as a type to declare an **object reference variable**
- **The object itself must be created separately**

Creating Objects

- We have already seen something like this:
  ```java
  Scanner scan = new Scanner (System.in);
  ```

  - The **new** operator calls the Scanner **constructor**, which is a special method that sets up the object
  - Variable refers to a **Scanner object**
  - Constructing a new object is called **instantiation**

String title = new String ("Java Software Solutions");

- **The String Class is SPECIAL!**
  - Exception to the use of **new** operator: Because strings are so common, we don't have to use the **new** operator to create a **String object**
    ```java
    String title = new String ("Java Software Solutions");
    ```
    ```java
    String title = "Java Software Solutions";
    ```
  - This is special syntax that works only for strings
Wrapper classes

- Wrapper classes
  - `Integer`, `Double`, `Char`, etc
  - Useful constants, eg, `Integer.MAX_VALUE`
  - Create objects of corresponding type (learn about this later)
  - Static methods to convert between types, eg:
    - `Double.parseDouble("3.14")`
    - `Integer.parseInt("54")`
    - etc

```
System.out.print("Enter account number");
String line = scan.nextLine();  // eg: 23 88 24
noSpaces = line.replaceAll(" ","");  // remove spaces
int number = Integer.parseInt(noSpaces);  // store as int
```

More Java Classes

- Formatting
  - `NumberFormat`
  - `DecimalFormat`
  - many others
- Text processing
- Web development
- 3D Graphics
- Animation
- Scientific applications
- Multi-precision arithmetic
- Vendor specific APIs (eg Twitter or Facebook)
- Graphical user interface development (next week)

... and Much, much more!

Data Representation

- Computers store all information **digitally**, using **binary** codes:
  - numbers
  - text
  - images
  - audio
  - video
  - program instructions

```
Opening JGrasp and click on the Interactions tab (lower part of window).
Type any expression to get its value; type variable names to get their values.
Watch the Workbench tab on the top/left part of the window; it lists your variables and their values.
You can type Java statements such as variable declarations, conditionals, loops and even import directives!
```
Why Binary Numbers?

- Simplest way to represent digital information:
  - Electronic circuits: high/low voltage
  - Magnetic devices (e.g., hard drive): positive/negative
  - Optical devices (e.g., DVD): light reflected/not reflected due to microscopic grooves

A binary digit is called a **bit** - *binary digit*

A **byte** is a group of eight bits

### Binary Codes

<table>
<thead>
<tr>
<th>1 bit</th>
<th>2 bits</th>
<th>3 bits</th>
<th>4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>011</td>
<td>0011</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>110</td>
<td>0100</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>111</td>
<td>0101</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>100</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>110</td>
<td>101</td>
<td>0111</td>
</tr>
<tr>
<td>1100</td>
<td>111</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td>1100</td>
<td>1001</td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td>1110</td>
<td>1010</td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td>1111</td>
<td>1100</td>
</tr>
</tbody>
</table>

Each additional bit doubles the number of possible codes

### Storage Capacity

- Every memory device has a **storage capacity**, indicating the number of bytes it can hold

<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobyte</td>
<td>KB</td>
<td>$2^{10} = 1024$</td>
</tr>
<tr>
<td>megabyte</td>
<td>MB</td>
<td>$2^{20}$ (over one million)</td>
</tr>
<tr>
<td>gigabyte</td>
<td>GB</td>
<td>$2^{30}$ (over one billion)</td>
</tr>
<tr>
<td>terabyte</td>
<td>TB</td>
<td>$2^{40}$ (over one trillion)</td>
</tr>
<tr>
<td>petabyte</td>
<td>PB</td>
<td>$2^{50}$ (a whole bunch)</td>
</tr>
</tbody>
</table>
**Numeric Primitive Data**

- The difference between the numeric primitive types is their size and the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^18</td>
<td>9 x 10^18</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^308 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

**Characters in Java**

- Characters, including spaces, digits, and punctuation are represented by numeric codes

```
Hi, Heather.
```

- `72` 105 44 32 101 97 116 104 101 114 46

**Characters in Java**

- A `char` variable stores a single character
- Character literals are delimited by single quotes:
  ```
  'a' 'X' '7' '§' ',' '
  ```

```java
char grade = 'A';
char terminator = ' '; separator = ' ', newline = '\n';
String oneLetter = "A";  // this NOT the same
```

**Automatic type conversion**

Values of different types can be combined in an assignment or an expression

- **Example:**

  ```java
  int dollars = 2;
  double money = dollars + 0.50;
  System.out.println(dollars + " dollars");
  ```

- These are all examples of **widening conversions**, i.e., “smaller” data type ➔ “larger” data type
Converting from one type to another

• **Widening conversions**
  – "small" data type → "larger" one
  • eg: int → double
    32 bits → 64 bits

• **Narrowing conversions**
  – "large" data type → "smaller" one
  • eg: double → int
    64 bits → 32 bits
  – narrowing conversions can lose information!
  – narrowing conversions cannot happen automatically (for example, through assignment)

Casting

• *Casting* forces a change of type, even if information is lost
• Can be used for both widening and narrowing conversion
• To cast, put the type in parentheses in front of the value to be converted:

```java
int total = 5;
double result = (double) total / 2;
int answer = (int) result + 4;
double angle = Math.PI / 8;
int x = (int) (Math.cos(angle) * 300);
```

(char ↔ int) Conversion

• A char variable is stored as its unicode representation
• char ↔ int conversion: convert between char and its unicode, eg:
  ```java
  (int) 't' ↔ 116
  ```
• increment and decrement of char variables takes you up and down in alphabetical order (codes are in numeric sequence), eg:
  ```java
  char letter = 'B';
  letter ++ ↔ 'C'
  ```

Try this:

Random gen = new Random();

// randomly generated letter

// randomly generated three-letter code
Casts – try these examples

(double) 4 / 3 ______ (double) (4 / 3) ______
(int) (0.7 * 0.7 * 100) ______

Data Conversion

How to use cast?

Forcing floating point division between int expressions

```java
int qp = 35;
int credits = 10;
double gpa = (double) qp / credits;
gpa should be 3.5
```

```java
int qp = 35;
int credits = 10;
double gpa = (double) (qp / credits);
```

Scaling a double and converting to int

```java
double gpa = 3.2;
int gpaPercent = (int) ((gpa / 4) * 100);
gpaPercent should be 80
```

```java
double gpa = 3.2;
int gpaPercent = (int) ((gpa / 4) * 100);
```