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/

Overview

- The Java API classes
  - Math class
  - Random class
  - String class
  - Learning about classes in the Java API
- Binary representation
- Data types revisited
- Type conversions, casts

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

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</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>java.text</td>
<td>Text utilities (e.g., formatting)</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>javafx.scene</td>
<td>Graphical shapes</td>
</tr>
<tr>
<td>javafx.scene.control</td>
<td>GUI controls</td>
</tr>
</tbody>
</table>

Some slides in this presentation are adapted from the slides accompanying Java Software Solutions by Lewis & Loftus.
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 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);
  ```

---

Some methods from the Math class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Math.abs(a)</code></td>
<td>absolute value of <code>a</code></td>
</tr>
<tr>
<td><code>Math.max(a, b)</code></td>
<td>maximum of <code>a</code> and <code>b</code></td>
</tr>
<tr>
<td><code>Math.min(a, b)</code></td>
<td>minimum of <code>a</code> and <code>b</code></td>
</tr>
<tr>
<td><code>Math.sin(theta)</code></td>
<td>sine function</td>
</tr>
<tr>
<td><code>Math.cos(theta)</code></td>
<td>cosine function</td>
</tr>
<tr>
<td><code>Math.tan(theta)</code></td>
<td>tangent function</td>
</tr>
<tr>
<td><code>Math.exp(a)</code></td>
<td>exponential (e)</td>
</tr>
<tr>
<td><code>Math.log(a)</code></td>
<td>natural log (ln <code>a</code>)</td>
</tr>
<tr>
<td><code>Math.pow(a, b)</code></td>
<td>raise <code>a</code> to the <code>b</code> power</td>
</tr>
<tr>
<td><code>Math.round(a)</code></td>
<td>round to the nearest integer</td>
</tr>
<tr>
<td><code>Math.random()</code></td>
<td>random number in [0, 1)</td>
</tr>
<tr>
<td><code>Math.sqrt(a)</code></td>
<td>square root of <code>a</code></td>
</tr>
</tbody>
</table>

---

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]
  ```

---

[CSC 1051 M.A. Papalaskari, Villanova University](#)
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) // snake eyes
        count++; // snake eyes
    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 [0.0,1.0), 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 + ");
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
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?
Data Representation and Applets

System.out.println("Enter a potential palindrome: ");
str = scan.nextLine();
left = 0;
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");

Creating Objects
• We have already seen something like this:

Creating Objects
• Another example:

Declaring Variables, revisited
• Examples of variable declarations:
  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
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");
```

- This is special syntax that works only for strings

```java
String title = "Java Software Solutions";
```

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

```java
System.out.print("Enter account number");
String line = scan.nextLine(); // eq: 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 – warmup exercise

1 / 200 _________ 1.0 / 200 _________
1.0 / 2000 _________ 1.0 / 20000 _________
1 + 1 == 3 _________
0.7 + 0.7 == 1.4 _________
0.7 * 0.7 == .49 _________
0.7 * 0.7 - .49 _________

Data Representation and Applets
Data Representation

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

Why Binary Numbers?

- Simplest way to represent digital information:
  - Electronic circuits: high/low voltage
  - Magnetic devices (eg hard drive): positive/negative
  - Optical devices (eg 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

- How many codes?
- 1 bit?
- 2 bits?
- 3 bits?
- 4 bits?
- 5 bits?

- How many codes of N bits?
- How many bits are needed to represent 64 items?
- How many bits are needed to represent 1000 items?
Storage requirements examples

- A code requires 16 bits (e.g., two bytes, representing an letter of the English alphabet) a document consisting of 1000 such codes will require _____ bits or _____ bytes.

- A code requires 32 bits (e.g., a pixel in an image), a program that needs to store a 1000x1000 grid of such codes will require _____ bits or _____ bytes.

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 \times 10^{15})</td>
<td>(&gt; 9 \times 10^{15})</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 \times 10^{38} with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 \times 10^{30} with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

Characters in Java

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

\[H\]i, \textit{H}e\textit{a}t\textit{h}e\textit{r}.

The ASCII (American Standard Code for Information Interchange) character set uses eight bits per character, allowing for 256 unique characters.

The Unicode character set extends ASCII to sixteen bits per character, allowing for 65,536 unique characters.
Characters in Java

• A `char` variable stores a single character
• Character literals are delimited by single quotes:
  `'a'   'X'    '7'    '$'    ','    '\n'``

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

Note the difference between a primitive `char` variable, which holds only one character, and a `String` object, which can hold multiple characters.

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);
```

(cast has higher precedence than arithmetic operators)
**char ⇔ int Conversion**

- A `char` variable is stored as its unicode representation
- `char ⇔ int` conversion: convert between char and its unicode, eg:
  - `(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:**

```java
Random gen = new Random();
// randomly generated letter

// randomly generated three-letter code
```

**Casts – try these examples**

```java
(double) 4 / 3 _____ (double) (4 / 3) _____
(int) (0.7 * 0.7 * 100) _________
```

**Data Conversion**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>short, int, long, float, or double</td>
</tr>
<tr>
<td>short</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>char</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>byte</td>
<td>short, or char</td>
</tr>
<tr>
<td>char</td>
<td>byte or short</td>
</tr>
<tr>
<td>short</td>
<td>byte, short, or char</td>
</tr>
<tr>
<td>int</td>
<td>byte, short, or char</td>
</tr>
<tr>
<td>long</td>
<td>byte, short, or char</td>
</tr>
<tr>
<td>float</td>
<td>byte, short, char, int, or long</td>
</tr>
<tr>
<td>double</td>
<td>byte, short, char, int, long, or float</td>
</tr>
</tbody>
</table>
**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);
```

**How to use cast?**

*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);
```