**Data Representation Classes, and the Java API**

**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**

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

---

**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

1 bit  2 bits  3 bits  4 bits
0     00    000    0000
1     01    001    0001
10    010   0011   0010
11    011   0100   0101
100   100   0110   1010
101   110   1011   1000
110   111   1110   1011
111

Each additional bit doubles the number of possible codes.

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 80 items?
• How many bits are needed to represent each of the 50 states (so that each state corresponds to a unique code)?

Storage requirements examples

• If a code requires 5 bits, a document consisting of 4000 such codes will require a total of:
  \[5 \times 4000 = 20,000 \text{ bits}\]
  – how many bytes is that? _______________

• If a code requires 8 bits (i.e., a byte), a document consisting of 2000 such codes will require 2000 bytes.
  – how many bits is that? _______________

• If a code requires 32 bits, a program that needs to store 2000 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} \text{ (over one million)})</td>
</tr>
<tr>
<td>gigabyte</td>
<td>GB</td>
<td>(2^{30} \text{ (over one billion)})</td>
</tr>
<tr>
<td>terabyte</td>
<td>TB</td>
<td>(2^{40} \text{ (over one trillion)})</td>
</tr>
<tr>
<td>petabyte</td>
<td>PB</td>
<td>(2^{50} \text{ (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>&gt; 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

- A `char` variable stores a single character.
- Character literals are delimited by single quotes:

  `'a'  'X'  '7'  '$'  ','  '"'  '
'

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

```java
char grade = 'A';
char terminator = ';'; separator = ' ', newline = '\n';
char letter = 't';
char next = letter++; // 'u'
```

### Characters in Java

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

```
    Hi, Heather.
```

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.

### Automatic type conversion

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

- **Example:**

  ```java
  int dollars = 5;
  double money = dollars + 2.50;
  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)

### Example
```java
int total = 5;
double result = (double) total / 2;
int answer = (int) result + 4;
```

### 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;
```

### Data Conversion

<table>
<thead>
<tr>
<th>Widening Conversions</th>
<th>Narrowing Conversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>To</td>
</tr>
<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>int</td>
<td>long, float, or double</td>
</tr>
<tr>
<td>long</td>
<td>float or double</td>
</tr>
<tr>
<td>Float</td>
<td>double</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`

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

`gpaPercent` should be 80

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

char <-> int Conversion

• A char variable is stored as its unicode representation
• char <-> int conversion: convert between char and its unicode.
• e.g.: ‘t’ <-> 116
• Increment and decrement of char variables takes you up and down in alphabetical order (codes are in numeric sequence)

```
char letter = 't';
int letterCode = letter;  // 116 (code for 't')
char next = letter++;    // 'u'
int nextCode = next;     // 117 (code for 'u')
char nextAlt = (char) nextCode;  // 'u'
// narrowing conversion - requires cast
char whatsthis = (char) ('A' + 3);  // ?
int num = (int) (letter - 'p');  // ??
```

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

- 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.shape</td>
<td>Graphical shapes</td>
</tr>
<tr>
<td>javafx.scene.control</td>
<td>GUI controls</td>
</tr>
</tbody>
</table>
```

Packages

Copyright © 2017 Pearson Education, Inc.
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

- `abs(double a)` absolute value of `a`
- `max(double a, double b)` maximum of `a` and `b`
- `min(double a, double b)` minimum of `a` and `b`
- `sin(double theta)` sine function
- `cos(double theta)` cosine function
- `tan(double theta)` tangent function
- `exp(double a)` exponential (e^a)
- `log(double a)` natural log (log_e(a), or ln a)
- `pow(double a, double b)` raise `a` to the `b`th power (a^b)
- `random()` random number in [0, 1)
- `round(double a)` round to the nearest integer
- `sqrt(double a)` square root of `a`
- `E` value of e (constant)
- `PI` value of π (constant)

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);
  ```
  ```java
  // 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

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

Math class method

Try this:
1) Randomly generated 4-digit code
2) Randomly generated 4-letter code

Random gen = new Random();

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++; // snake eyes
    trial++;
}
System.out.println ("Probability of snake eyes = " +
        (double)count/100000);
Monte Carlo simulation example: approximate the value of \( \pi \)

```java
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
    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:

```
String name = "Betsy";

int numOfCharsInName = name.length();
```

More String Methods

- `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?


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**, an instance of the Scanner class
Creating Objects

- Another example:

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

  The `new` operator calls the String constructor, which is a special method that sets up the object.

  Variable refers to a String object

  Constructing a new object is called instantiation.

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

```java
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!