Joanna Klukowska
joannakl@cs.nyu.edu

Lecture 2:
Words and Numbers

Contents

1 Math in Java **2**
1.1 `Math` class, or your mathematical toolbox . . . 2
1.2 Random Number Generator Using `Math.random()` . . . 2
1.2.1 Pseudo-random ⇔ Random . . . 2
1.2.2 How to Generate Random Numbers and Characters . . . 2
1.3 Minimizing numerical errors . . . 2

2 Characters in Java **3**
2.1 Characters are just numbers . . . 3
2.2 `Character` class, or your character toolbox . . . 3
2.3 Escape sequences . . . 3

3 Strings (Words) in Java **4**
3.1 String class, or your string toolbox . . . 4

Joanna Klukowska
joannakl@cs.nyu.edu

1 Math in Java

1.1 `Math` class, or your mathematical toolbox

The `Math` class in Java provides many tools that allow us to perform mathematical operations within the program. You have already seen some of the methods in it: `Math.sqrt(...)` and `Math.pow(..., ...)`. The documentation page for `Math` class provides descriptions of all of the available methods: http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html. Most of them have very intuitive names (matching their mathematical equivalents) so they do not need much explanation.

Here are just a few examples of what `Math` class has to offer

`Math.cos()`	`Math.atan()`	`Math.log10()`	`Math.round()`
`Math.sin()`	`Math.toRadians()`	`Math.pow()`	`Math.min()`
`Math.tan()`	`Math.toDegrees()`	`Math.sqrt()`	`Math.max()`
`Math.asin()`	`Math.exp()`	`Math.ceil()`	`Math.abs()`
`Math.acos()`	`Math.log()`	`Math.floor()`	`Math.random()`

1.2 Random Number Generator Using `Math.random()`

Math class method `Math.random()` returns a pseudo-random double value between 0.0 (inclusive) and 1.0 (exclusive). or the detailed documentation see http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#random--

1.2.1 Pseudo-random $\Leftrightarrow$ Random

Nothing generated by a computer can be truly random. There is an algorithm and code that does it. But good pseudo-random number generators appear to be random and offer statistical properties that are like the ones of the truly random things.

There have been some attempts of generating truly random numbers based on physical properties. See this Wikipedia page for an example: http://en.wikipedia.org/wiki/Lavarand.

1.2.2 How to Generate Random Numbers and Characters

Since Math.random() generates numbers only in the range of $[0, 1)$ we need to do some shifting and scaling to produce pseudo-random real numbers in arbitrary ranges:

`a + Math.random()* b` returns a pseudo-random number between **`a`** and **`(a+b)`**, excluding **`(a+b)`**

In order to generate pseudo-random integers, you can still use Math.random(), but the result needs to be cast to and integer:

`(int)(a + Math.random()* b)` returns a pseudo-random number between **`a`** and **`(a+b-1)`**, assuming that both **`a`** and **`b`** are integers

Once you can generate pseudo-random positive integers (just keep **`a=0`** in the above formula), you can easily generate pseudo-random characters by casting the values to a **`char`** type:

`(char)((int)(Math.random()* b))` returns a pseudo-random character with random UTF-8 value of **`(int)(Math.random()* b)`**, assuming that **`b`** is an integer.

1.3 Minimizing numerical errors

The representation of floating point numbers on a computer is rarely accurate. Many fractions that have finite representation in the decimal system cannot be represented in finitely many binary symbols.

How would you compute the area and the perimeter of a rectangle with side lengths 4.5 and 7.9? See the code that implements it in `Rectangle.java`.

Joanna Klukowska
joannakl@cs.nyu.edu

2 Characters in Java

A character data type `char` represents a single Unicode character (UTF-8) to be exact.

ASCII (American Standard Code for Information Interchange) characters are a subset of **Unicode** characters with values from 0 to 127. There are 65,536 possible characters in Unicode (unextended).

How do these numbers correspond to alpha-numeric characters? Each letter, symbol, digit, etc. is mapped to a specific number in Unicode (for a listing of values corresponding to the ASCII subset see Appendix B in your textbook). For a website that has all UTF-8 characters in many (many many) tables, see http://www.utf8-chartable.de/.

Using Unicode we can print all the "standard" characters from the keyboard, but also things like chess symbols, characters of other languages, etc.

2.1 Characters are just numbers

Each character in Unicode has a unique number assigned to it. Within a Java programs we can move between the numbers and characters using casting (temporarily interpreting a value as a different type.

```
char letter1 = 'A';
int number1 = 90;

System.out.println("Character " + letter1 + " has code " + (int)letter1 );
System.out.println("Character " + (char)number1 + " has code " + number1 );
```

prints

Output:

```
Character A has code 65 }
Character Z has code 90 }
```

2.2 `Character` class, or your character toolbox

The **`Character`** class http://docs.oracle.com/javase/8/docs/api/java/lang/Character.html provides many tools that allow us to manipulate and classify characters. Here are just a few examples of methods from the **`Character`** class:

```
Character.isDigit( c )
Character.isLetter( c )
Character.isLetterOrDigit( c )
Character.isLowerCase( c )
Character.isUpperCase( c )
Character.toLowerCase( c )
Character.toUpperCase( c )
```

To compare characters we use the **`<, >, <=, >=, ==`** operators. The result of comparison depends on the numerical value associated with the characters.

2.3 Escape sequences

Due to the rules of Java and a nature of some characters (such as tab, new line, ...) certain characters are hard to print. We can use the UTF-8 codes to print them, but they are hard to remember. There are some characters that can be printed using escape sequences. An escape sequence is a character preceded by a backslash. The following list shows all escape sequences in Java and their meaning.

\b Backspace

\t Tab

Joanna Klukowska
joannakl@cs.nyu.edu

\n Line feed (or a new line)

\r Carriage return

\f Form feed

\\ Backslash

\" Double quote

3 Strings (Words) in Java

A **string** is a sequence of characters. In Java we use `String` type/class to represent strings in the code. Java's `String` is not a primitive type.

```
String message = "Welcome to Java";
```

You can use the + operator to **concatenate** strings:

```
String part1 = "Welcome";
String part2 = " to Java";
String message = part1 + part2;
System.out.println(message);
```

prints "Welcome to Java" on the screen.

Reading strings from the user. You can use `next()` method of the `Scanner` class to read strings from the user:

```
Scanner in = new Scanner(System.in);
String name;
System.out.println ("Enter you name: ");
name = input.next();
```

3.1 String class, or your string toolbox

The String class comes with many tools that make it easier to manipulate strings. For the list of them see http://docs.oracle.com/javase/7/docs/api/java/lang/String.html. For not, you should be familiar with how to do the following things:

Getting string length You can use a method called `lenght()` to obtain the length of the string. For example:

```
String s = "Hello";
System.out.print(s + " has " + s.length() + " letters.");
```

prints "Hello has 5 letters."

Getting individual characters from the string Think of letters in the string as individual characters numbered from left to right starting at zero. String "Hello" is numbered as follows:

```
H | e | l | l | o
0 | 1 | 2 | 3 | 4
```

The method `charAt( numberOfCharacter )` gives us the individual characters. For example:

```
String s = "Hello";
System.out.println("First character is: " + s.get(0) );
System.out.println("Third character is: " + s.get(2) );
System.out.println("Last character is: " + s.get(s.length() -1 ) );
```

Joanna Klukowska
joannakl@cs.nyu.edu

prints

Output:

```
First character is H
Third character is l
Last character is o
```

Comparing content of two strings Remember that we use the `<, >, <=, >=, ==` operators to compare numbers and characters. To compare strings we need to use other tools.

Use `equals( ... )` to check if two strings are identical. For example:

```
String s1 = "Hello";
String s2 = "hello";
if ( s1.equals(s2) )
  System.out.println("s1 and s2 are the same");
else
  System.out.println("s1 and s2 are different");
```

prints `s1 and s2 are different`.

If you wish to ignore the case of letters in the comparison, use `equalsIgnoreCase(...)`. Using `s1` and `s2` from the previous example

```
if ( s1.equalsIgnoreCse(s2) )
  System.out.println("s1 and s2 are the same");
else
  System.out.println("s1 and s2 are different");
```

prints `s1 and s2 are the same`.

To compare the Unicode (or alphanumeric) ordering of two strings, use `compareTo(...)` or `compareToIgnoreCase(...)`. We will revisit those methods later on in the course.