Transitioning to Java II: Strings, Files, and Functions
Strings
Strings: Example Program

```java
public class Ruler {
    public static void main(String[] args) {
        String ruler1 = "1";
        String ruler2 = ruler1 + " 2 " + ruler1;
        String ruler3 = ruler2 + " 3 " + ruler2;
        String ruler4 = ruler3 + " 4 " + ruler3;
        System.out.println(ruler4);
    }
}
```

Download `Ruler.java` from booksite, section 1.2
Strings

• Strings are really arrays of chars

```java
String name = "M. Smith";
System.out.println(name.charAt(3));  // s
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>M</td>
<td>.</td>
<td>S</td>
<td>m</td>
<td>i</td>
<td>t</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

• Once a String is created it cannot be changed
  • Strings are *immutable*
  • Therefore, we must reassign it. For example:
    ```java
    String s = "how i met your mother";
    s = s.toUpperCase();
    System.out.println(s);  // HOW I MET YOUR MOTHER
    ```
String methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexOf(str)</td>
<td>index where the start of the given string appears in this string (-1 if it is not there)</td>
</tr>
<tr>
<td>length()</td>
<td>number of characters in this string</td>
</tr>
<tr>
<td>substring(index1, index2) or substring(index1)</td>
<td>the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 omitted, grabs till end of string</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>a new string with all lowercase letters</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>a new string with all uppercase letters</td>
</tr>
</tbody>
</table>

- These methods are called using the dot notation:

```java
String test = "Hello world";
System.out.println(test.length()); // 11
```
String Method Examples

```java
// 01234567890123
String s1 = "Hello CIS 110!";
System.out.println(s1.length()); // 14
System.out.println(s1.indexOf("e")); // 1
System.out.println(s1.substring(6, 9)) // "CIS"

String s2 = s1.substring(6, 13);
System.out.println(s2.toLowerCase()); // "cis 110"
```

- How would you extract the first word from ANY string?
Strings as parameters

```java
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Lily");

        String lawyer = "Marshall";
        sayHello(lawyer);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}

Output:
Welcome, Lily
Welcome, Marshall
```
Comparing Strings

- Relational operators such as `<` and `==` fail on Strings.

```java
public static void main(String[] args) {
    String name = args[0];
    if (name == "Barney") {
        System.out.println("It’s gonna be legend- (wait for it)!");
        System.out.println("-dary!");
    }
}
```

- This code will compile, but it will not print the phrase.

- `==` compares objects by references, so it often gives false even when two Strings have the same letters.
Comparing Strings

- Objects are compared using a method named `equals`.

```java
public static void main(String[] args) {
    String name = args[0];
    if (name.equals("Barney")) {
        System.out.println("It’s gonna be legend- (wait for it)");
        System.out.println("-dary!");
    }
}
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals(str)</code></td>
<td>whether two strings contain the same characters</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(str)</code></td>
<td>whether two strings contain the same characters, ignoring upper vs. lower case</td>
</tr>
<tr>
<td><code>startsWith(str)</code></td>
<td>whether one contains other's characters at start</td>
</tr>
<tr>
<td><code>endsWith(str)</code></td>
<td>whether one contains other's characters at end</td>
</tr>
<tr>
<td><code>contains(str)</code></td>
<td>whether the given string is found within this one</td>
</tr>
</tbody>
</table>
String/char question

- A Caesar cipher is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
Caesar Cipher 1

// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

public class SecretMessage {
    public static void main(String[] args) {

        System.out.print("Your secret message: ");
        String message = StdIn.readLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = StdIn.readInt();

        encode(message, key);
    }

   ...

Based on slides by S. Reges and M. Stepp, Building Java Programs, Pearson Ed.
Caesar Cipher 2

// This method encodes the given text string using a Caesar cipher, shifting each letter by the given number of places.
public static void encode(String text, int shift) {
    System.out.print("The encoded message: ");
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        // shift only letters (leave other chars alone)
        if (letter >= 'a' && letter <= 'z') {
            letter = (char) (letter + shift);
        } else if (letter < 'a') {
            letter = (char) (letter + 26);
        } else if (letter > 'z') {
            letter = (char) (letter - 26);
        }

        // may need to wrap around
        if (letter > 'z') {
            letter = (char) (letter - 26);
        } else if (letter < 'a') {
            letter = (char) (letter + 26);
        }

        System.out.print(letter);
    }
}
}
## Self-Test: Strings

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;This is a string literal.&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot; + &quot;2&quot;</td>
<td></td>
</tr>
<tr>
<td>1 + &quot; +&quot; + 2 + &quot; = &quot; + 3</td>
<td></td>
</tr>
<tr>
<td>'1' + &quot;2&quot;</td>
<td></td>
</tr>
<tr>
<td>0 + '1' + &quot;2&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;&quot; + Math.sqrt(2)</td>
<td></td>
</tr>
<tr>
<td>(String) Math.sqrt(2)</td>
<td></td>
</tr>
<tr>
<td>(string) Math.sqrt(2)</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot; == &quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;.equals(&quot;A&quot;)</td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot; &lt; &quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot;.compareTo(&quot;A&quot;)</td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot;.compareTo(&quot;B&quot;)</td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot;.compareTo(&quot;C&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
Files
Text Files

- Text files are just sequences of characters

- Example file format:  
  
<table>
<thead>
<tr>
<th>1</th>
<th>3.4</th>
<th>JSmith</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.9</td>
<td>MJones</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Can read using the `In` library (provided with your textbook)

  ```java
  In inStream = new In("data.txt");
  while (!inStream.isEmpty()) {
      int idx = inStream.readInt();
      double gpa = inStream.readDouble();
      String name = inStream.readString();
      ...
  }
  ```
Functions
Method Overloading

• Two or more methods *in the same class* may also have the same name

• This is called *method overloading*

```java
// absolute value of an int value
public static int abs(int x)
{
    if (x < 0) return -x;
    else return x;
}

// absolute value of a double value
public static double abs(double x)
{
    if (x < 0.0) return -x;
    else return x;
}
```
Method Signature

• A method is uniquely identified by
  – its **name** and
  – **its parameter list** (parameter types and their order)
• This is known as its **signature**

Examples:

static int min (int a, int b)
static double min (double a, double b)
static float min (float a, float b)
Return Type is Not Enough

• Suppose we attempt to create an overloaded `circle(double x, double y, double r)` method by using different return types:

```java
static void circle (double x, double y, double r) {...}
//returns true if circle is entirely onscreen, false otherwise
static boolean circle (double x, double y, double r) {...}
```

• This is NOT valid method overloading because the code that calls the function can ignore the return value

```
circle(50, 50, 10);
```

– The compiler can’t tell which `circle()` method to invoke
– Just because a method returns a value doesn’t mean the calling code has to use it
Too Much of a Good Thing

Automatic type promotion and overloading can sometimes interact in ways that confuse the compiler. For example:

```
//version 1
static void printAverage (int a, double b) {
  ...
}

//version 2
static void printAverage (double a, int b) {
  ...
}
```

Why might this be problematic?
Too Much of a Good Thing

static void printAverage (int a, double b) {/*code*/}
static void printAverage (double a, int b) {/*code*/}

• Consider if we do this:

```java
public static void main (String[] args) {
    ...
    printAverage(4, 8);
    ...
}
```

• The Java compiler can’t decide whether to:
  – promote 7 to 7.0 and invoke the first version of printAverage(), or
  – promote 5 to 5.0 and invoke the second version
• It will throw up its hands and complain
• Take-home lesson: don’t be too clever with method overloading
More Documentation
Method-level Documentation

• Method header format:

```java
/**
 * Name: circleArea
 * PreCondition: the radius is greater than zero
 * PostCondition: none
 * @param radius - the radius of the circle
 * @return the calculated area of the circle
 */
static double circleArea (double radius) {
    // handle unmet precondition
    if (radius < 0.0) {
        return 0.0;
    } else {
        return Math.PI * radius * radius;
    }
}
```
Method Documentation

• Clear communication with the class user is of paramount importance so that he can
  – use the appropriate method, and
  – use class methods properly.

• Method comments:
  – explain what the method does, and
  – describe how to use the method.

• Two important types of method comments:
  – precondition comments
  – post-conditions comments
Preconditions and Postconditions

• **Precondition**
  – What is assumed to be true when a method is called
  – If any pre-condition is not met, the method may not correctly perform its function.

• **Postcondition**
  – States what will be true after the method executes (assuming all pre-conditions are met)
  – Describes the side-effect of the method
An Example

Very often the precondition specifies the limits of the parameters and the postcondition says something about the return value.

/* Prints the specified date in a long format
   e.g. 1/1/2000 -> January 1, 2000
   Pre-condition:
       1 <= month <= 12
       day appropriate for the month
       1000 <= year <= 9999
   Post-condition:
       Prints the date in long format
   */
static printDate(int month, int day, int year)
{
    // code here
}
## Function Examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Code Example</th>
</tr>
</thead>
</table>
| **absolute value of an int value** | `public static int abs(int x) {
    if (x < 0) return -x;
    else return x;
}`                                                                        |
| **absolute value of a double value** | `public static double abs(double x) {
    if (x < 0.0) return -x;
    else return x;
}`                                                                      |
| **primality test**                 | `public static boolean isPrime(int N) {
    if (N < 2) return false;
    for (int i = 2; i <= N/i; i++)
        if (N % i == 0) return false;
    return true;
}`                                                                  |
| **hypotenuse of a right triangle** | `public static double hypotenuse(double a, double b) {
    return Math.sqrt(a*a + b*b);
}`                                                                      |
Function Challenge 1a

Q. What happens when you compile and run the following code?

```java
public class Cubes1 {
    public static int cube(int i) {
        int j = i * i * i;
        return j;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + " " + cube(i));
    }
}
```

% `javac Cubes1.java`
% `java Cubes1 6`
1 1
2 8
3 27
4 64
5 125
6 216
Function Challenge 1b

Q. What happens when you compile and run the following code?

```java
public class Cubes2 {
    public static int cube(int i) {
        int i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + " " + cube(i));
    }
}
```
Function Challenge 1c

Q. What happens when you compile and run the following code?

```
public class Cubes3 {
    public static int cube(int i) {
        i = i * i * i;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + " " + cube(i));
    }
}
```
Function Challenge 1d

Q. What happens when you compile and run the following code?

```java
public class Cubes4 {
    public static int cube(int i) {
        i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + " " + cube(i));
    }
}
```
Function Challenge 1e

Q. What happens when you compile and run the following code?

```java
public class Cubes5 {
    public static int cube(int i) {
        return i * i * i;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + " " + cube(i));
    }
}
```