Strings
& their Methods

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 3.3 & 4.3
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/) & thanks to Ms Martin.
Let’s review the Types of Variables we have used:

**Primitive Types:**
- `int name = <value>`; // create an Integer
- `double name = <value>`; // create an Double — real numbers
- `char name = ‘<single character>’`; // create a single character
- `boolean name = true`; // creates a Boolean of true or false.

**Object Type:**
- `String name = “<series of characters>”`; // creates a String of char’s
Objects

- **object**: An entity that contains data and behavior.
  - *data*: variables inside the object
  - *behavior*: methods inside the object
  - You interact with the methods; the data is hidden in the object.

- Constructing (creating) an object:
  ```
  Type objectName = new Type(parameters);
  ```

- Calling an object's method:
  ```
  objectName . methodName (parameters);
  ```
Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a String is not created with `new`.

  ```java
  String name = "text";
  String name = expression;
  ```

- Examples:

  ```java
  String name = "Marla Singer";
  int x = 3;
  int y = 5;
  String point = "(" + x + ", " + y + ")";
  ```
Characters of a string are numbered with 0-based *indexes*:

String name = "R. Kelly";

<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>character</td>
<td>R</td>
<td>.</td>
<td>K</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

- First character's index : 0
- Last character's index : 1 less than the string's length
- The individual characters are values of type *char* (seen later)
### String methods

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

- These methods are called using the dot notation:

```java
String gangsta = "Dr. Dre";
System.out.println(gangsta.length()); // 7
```
String method examples

// index    012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";

System.out.println(s1.length());    // 12
System.out.println(s1.indexOf("e")); // 8
System.out.println(s1.substring(7, 10)); // "Reg"

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase()); // "arty s"

• Given the following string:

// index    0123456789012345678901
String book = "Building Java Programs";

− How would you extract the word "Java" ?
Modifying strings

• Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```java
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);  // lil bow wow
```

• To modify a variable's value, you must reassign it:

```java
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);  // LIL BOW WOW
```
Strings as user input

• **Scanner's next method reads a word of input as a String.**

    Scanner console = new Scanner(System.in);
    System.out.print("What is your name? ");
    String name = console.next();
    name = name.toUpperCase();
    System.out.println(name + " has " + name.length() + " letters and starts with " + name.substring(0, 1));

    **Output:**
    What is your name? **Chamillionaire**
    Chamillionaire has 14 letters and starts with C

• **The nextLine method reads a line of input as a String.**

    System.out.print("What is your address? ");
    String address = console.nextLine();
Write a program that outputs a person's "gangsta name."
- first initial
- *Diddy*
- last name (all caps)
- first name
- -izzle

Example Output:
Type your name, playa: **Marge Simpson**
Your gangsta name is "M. Diddy SIMPSON Marge-izzle"
/ This program prints your "gangsta" name.
import java.util.*;

public class GangstaName {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type your name, playa: ");
        String name = console.nextLine();

        // split name into first/last name and initials
        String first = name.substring(0, name.indexOf(" "));
        String last = name.substring(name.indexOf(" ") + 1);
        last = last.toUpperCase();
        String fInitial = first.substring(0, 1);

        System.out.println("Your gangsta name is " + fInitial + ". Diddy " + last + " " + first + "-izzle");
    }
}
Comparing strings

• Relational operators such as < and == fail on objects.

Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me, ");
    System.out.println("We're a happy family!");
}

– This code will compile, but it will not print the song.

– == compares objects by references (seen later), so it often gives false even when two Strings have the same letters.
The *equals* method

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

```java
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technically this is a method that returns a value of type *boolean*, the type used in logical tests.
String name = console.next();
if (name.startsWith("Prof")) {
    System.out.println("When are your office hours?");
} else if (name.equalsIgnoreCase("STUART")) {
    System.out.println("Let's talk about meta!");
}
Type char

- **char**: A primitive type representing single characters.
  - A `String` is stored internally as an array of `char`

```java
String s = "Ali G.";
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>'A'</td>
<td>'l'</td>
<td>'i'</td>
<td>' '</td>
<td>'G'</td>
<td>'.'</td>
</tr>
</tbody>
</table>

- It is legal to have variables, parameters, returns of type `char`
  - surrounded with apostrophes: 'a' or '4' or '\n' or '\''

```java
char letter = 'P';
System.out.println(letter);  // P
System.out.println(letter + " Diddy");  // P Diddy
```
**The `charAt` method**

- The **characters** in a `String` can be accessed using the `charAt` method.
  - accepts an `int` index parameter and returns the `char` at that index

  ```java
  String food = "cookie";
  char firstLetter = food.charAt(0); // 'c'
  System.out.println(firstLetter + " is for " + food);
  
  String major = "CSE";
  for (int i = 0; i < major.length(); i++) {
    // output:
    char c = major.charAt(i);
    System.out.println(c);
    // C
    // S
    // E
  }
  ```
Comparing char values

- You can compare chars with ==, !=, and other operators:

```java
String word = console.next();
char last = word.charAt(word.length() - 1);
if (last == 's') {
    System.out.println(word + " is plural.");
}

// prints the alphabet
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```
char VS. int

• Each char is mapped to an integer value internally
  – Called an ASCII value

    'A' is 65  'B' is 66  ' ' is 32
    'a' is 97  'b' is 98  '*' is 42

  – Mixing char and int causes automatic conversion to int.
    'a' + 10 is 107,  'A' + 'A' is 130

  – To convert an int into the equivalent char, type-cast it.
    (char) ('a' + 2) is 'c'
char VS. String

- "h" is a `String`, but 'h' is a `char` (they are different)

- A `String` is an object; it contains methods.
  ```java
  String s = "h";
  s = s.toUpperCase(); // "H"
  int len = s.length(); // 1
  char first = s.charAt(0); // 'H'
  ```

- A `char` is primitive; you can't call methods on it.
  ```java
  char c = 'h';
  c = c.toUpperCase(); // ERROR
  s = s.charAt(0).toUpperCase(); // ERROR
  ```

- What is `s + 1`? What is `c + 1`?
- What is `s + s`? What is `c + c`?