Lecture 8: The String Class and Boolean Zen

Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp

Strings

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

```
Scanner input = new Scanner(System.in);
String name = "text";
String name2 = expression;
```

  - Examples:

```
String name = "Marla Singer";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
// (3, 5)
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of type `char`

`String` methods

Method name	Description
`indexOf(`**`str`**`)`	Returns index where the start of the given string appears in this string (-1 if not found)
`length()`	Returns number of characters in this string
`substring(`**`index1`**`, `**`index2`**`)` or `substring(`**`index1`**`)`	Returns the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end of string
`toLowerCase()`	Returns a new string with all lowercase letters
`toUpperCase()`	Returns a new string with all uppercase letters

- These methods are called using the dot notation:

```
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"
System.out.println(s1.substring(2));
// "uart Reges"
System.out.println(s1.substring(2,3));
// "u"
System.out.println(s1.substring(2,2));
// "" empty str
String s3 = s2.substring(1, 7);
// "arty S"
System.out.println(s3.toLowerCase());
// "arty s"
```

`String` method examples

- Given the following string:

```
// index          01234567890123456789
String book = "Building Java Programs";
```

  - How would you extract the word `"Java"` ?

```
String word=book.substring(9,13);
```

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.
  - String is **immutable**; once created, its value cannot be changed.

```
String s = ”kendrick";
s = “snoop dog”;
//”kendrick” is discarded and a new String
// object “snoop dog” is created.

s.toUpperCase();
System.out.println(s);
// snoop dog, s is not changed
```

Modifying strings

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

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

`String` test methods

Method	Description
`equals(`**`str`**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**`str`**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`contains(`**`str`**`)`	whether the given string is found within this one
`startsWith(`**`str`**`)`	whether the given string starts with this one
`endsWith(`**`str`**`)`	whether the given string ends with this one

```
if (name.contains("Prof")) {
    System.out.println("When are your office hours?");
}
```

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. The variable name and the literal string "Barney" are two different Strings even though they have the same characters.
  - `==` compares objects by *references* (seen later), so it often gives `false` even when two `String`s have the same letters.

The `equals` method

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

```
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.

Some String Properties

`String substring(int index1, int index2)`

–There is an IndexOutOfBoundsException if

a) index1 is negative

b) index2 is **larger** than the length of the string or

c) index1 is **larger** than index2.

```
String s="strawberry";
int x; String a;
x=s.length(); // 10
a=s.substring(5,9) // "berr"
a=s.substring(5,10) // "berry"
a=s.substring(5,11) // IndexOutOfBoundsException
a=s.substring(6,5) // IndexOutOfBoundsException
```

Some String Properties

`int indexOf(String str)`

–Returns index of first letter of first occurrence of str within this string. Returns -1 if not found. Error if str is null.

```
String a=“”; // a is empty string
String b; // b is null
String c=“happy”;
int x= c.indexOf(“ppy”); //x has value 2
x=c.indexOf(“hi”); //x now has value -1
x=c.indexOf(b); // error
x=c.indexOf(a); // x is 0
                // since e.g “” + “abc” = “abc”
```

Some String Properties

String substring(int index)

–There is an IndexOutOfBoundsException if index is negative or **larger** than the length of the string.

```
String s="cold",b;


b=s.substring(3); // b="d";

b="cold".substring(4); // b="", the empty string

b=s.substring(5); //IndexOutOfBoundsException

b="cold".substring(-3);//IndexOutOfBoundsException
```

Substring Methods

On the AP exam, only the following methods will be tested. Although there are a lot more useful methods in the String class, try to use only these in your programming projects.

```
length()
indexOf(String a)
substring(int index)
substring(int index1, int index2)
equals(String a)
```

More on `boolean`

"Boolean Zen", part 1

- Students new to `boolean` often test if a result is `true`:

```
if (isPrime(57) == true) {     // bad
    ...
}
```

- But this is unnecessary and redundant. Preferred:

```
if (isPrime(57)) {             // good
    ...
}
```

- A similar pattern can be used for a `false` test:

```
if (isPrime(57) == false) {    // bad
if (!isPrime(57)) {            // good
```

"Boolean Zen", part 2

- Methods that return `boolean` often have an `if/else` that returns `true` or `false`:

```
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```

  - But the code above is unnecessarily verbose.

Solution w/ `boolean` var

- We could store the result of the logical test.

```
public static boolean bothOdd(int n1, int n2) {
    boolean test = (n1 % 2 != 0 && n2 % 2 != 0);
    if (test) {    // test == true
        return true;
    } else {       // test == false
        return false;
    }
}
```

  - Notice: Whatever `test` is, we want to return that.
    - If `test` is `true` , we want to return `true`.
    - If `test` is `false`, we want to return `false`.

Solution w/ "Boolean Zen"

- Observation: The `if/else` is unnecessary.
  - The variable `test` stores a `boolean` value;
    its value is exactly what you want to return. So return that!

```
public static boolean bothOdd(int n1, int n2) {
    boolean test = (n1 % 2 != 0 && n2 % 2 != 0);
    return test;
}
```

- An even shorter version:
  - We don't even need the variable `test`.
    We can just perform the test and return its result in one step.

```
public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```

"Boolean Zen" template

- Replace

```
public static boolean name(parameters) {
    if (test) {
        return true;
    } else {
        return false;
    }
}
```

- with

```
public static boolean name(parameters) {
    return test;
}
```

isPrime method

```
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    if(factors==2)
        return true;
    else
        return false;

}
```

Improved `isPrime` method

- The following version utilizes Boolean Zen:

```
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    return factors == 2;  // if n has 2 factors, true
}
```

"Short-circuit" evaluation

- Java stops evaluating a test if it knows the answer.
  - `&&` stops early if any part of the test is `false`
  - `||` stops early if any part of the test is `true`

```
// Returns true if s1 and s2 end with the same two letters.
public static boolean rhyme(String s1, String s2) {
    return s1.endsWith(s2.substring(s2.length() - 2)) &&
           s1.length() >= 2 && s2.length() >= 2;
}
```

The test will crash if s2's length is less than 2.

"Short-circuit" fix

The following test will not crash; it stops if length < 2:

```
// Returns true if s1 and s2 end with the same two letters.
public static boolean rhyme(String s1, String s2) {
    return s1.length() >= 2 && s2.length() >= 2 &&
           s1.endsWith(s2.substring(s2.length() - 2));
}
```

De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negated Expression	Alternative
`a && b`	`!a || !b`	`!(a && b)`
`a || b`	`!a && !b`	`!(a || b)`

  - Example:

Original Code	Negated Code

`if (x == 7 && y > 3) {`
`  ...`
`}`
	`if (x != 7 || y <= 3) {`
	`  ...`
	`}`

De Morgan's Law

In Java:

```
!((age < 12) || (age >= 65))
```

In English: *It is not the case that age less than 12 or age greater than or equal to 65. !!!?*

Simplify using de Morgan's Law:

```
!(age < 12) && !(age >= 65)
```

The reverse the meaning of the relational expressions:

```
(age >= 12) && (age < 65)
```

That is, *when age is at least 12 and less than 65.*

Boolean practice questions

Write a method named `isVowel` that returns whether a `String` is a vowel (a, e, i, o, or u). Assume all letters are lowercase.

- `isVowel("q")` returns `false`
- `isVowel("a")` returns `true`
- `isVowel("e")` returns `true`

```
public static boolean isVowel(String s) {
    return s.equals("a") || s.equals("e") ||
           s.equals("i") || s.equals("o") ||
           s.equals("u");
}
```

Boolean practice questions

Change the above method into an `isNonVowel` method that returns whether a `String` is any character except a vowel.

- `isNonVowel("q")` returns `true`
- `isNonVowel("a")` returns `false`
- `isNonVowel("e")` returns `false`

What's the wrong strategy?

```
// Enlightened "Boolean Zen" version
public static boolean isNonVowel(String s) {
    return !s.equals("a") && !s.equals("e") &&
           !s.equals("i") && !s.equals("o") &&
           !s.equals("u");

}
```

Boolean practice questions

Use `isVowel` to write `isNonVowel`.

```
// Enlightened "Boolean Zen" version
public static boolean isNonVowel(String s) {
    return !isVowel(s);
}
```

Early Return

Early Return

```
//returns the sum of even integers from 1 to n.
  public static int sum(int n) {
    int sum=0;

    for(int i=1;i<=n;i++){
          if(i%2==0)
            sum+=i;
          return sum;
    }
}
```

Method returns too early. A return statement causes the method to immediately exit.

Fixed

```
//returns the sum of even integers from 1 to n.
  public static int sum(int n) {
    int sum=0;

    for(int i=1;i<=n;i++){
          if(i%2==0)
             sum+=i;
    }
    return sum;
}
```

return should be after the for loop.

Early Return?

Is this an example of early return?

```
//returns the first prime from m to n. If there are
// none, returns -1.
  public static int early(int m, int n) {

    for(int i=m;i<=n;i++){
          if(isPrime(i))
            return i;
    }
    return -1;
}
```

Method returns correctly.

Lab 1: Gangsta Name

- Write a method `gangstaName` which accepts a String input and returns a person's "gangsta name." Assume that the input name has only a first and last 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
"Why did Snoop Dog bring an umbrella?"
"For drizzle"
```

Lab 2

In this lab, you'll write 4 methods: `countSpaces`, `extract`, `hasVowel` and `consonant`.

- Write a void method `countSpaces` that takes a String parameter and return the number of spaces. **Write two versions of this method:** one uses a for loop and one uses a while loop. Use `indexOf`.

```
countSpaces("This line has four spaces."); // 4
```

Lab 2

Write a method `extract` which accepts a string parameter `str` and returns the string consists of all vowels in `str` in the same order. Use `isVowel`. Assume `str` is all lowercase.

Use a for loop to examine each individual string character.

`extract("programming")` returns `"oai"`.
`extract("bcd")` returns `""`.

Lab 2

Write a method `hasVowel` which accepts a String parameter `str` and returns whether the `str` contains any vowel. Use `isVowel`. Assume `str` is all lowercase.

Use a for loop to examine each individual string character.

```
hasVowel("ng");   //returns false
hasVowel("bbbacadabra"); // returns true
```

Lab 2

Write a method `consonant` which accepts a String parameter `str` and returns whether the `str` contains ONLY consonants. Use `isVowel`.

Use a for loop to examine each individual string character.

```
consonant("bcdfghjk"); //returns true
consonant("bcdefgh"); //returns false
hasVowel("bbbacadabra"); // returns true
```