Strings and String Operations

Overview

- Creating String Objects
- Substring methods
- The Concatenation Operator
- Strings are Immutable
- Other Methods of the String Class
- Example using String Methods
- Formatting Floating-point Numbers
- Preview: Console Input

Creating String Objects

Strings are Objects

- String is a sequence of characters enclosed in quotes. E.g. "Hello"
- String processing is one of the most important and most frequent applications of a computer
- Java recognizes this fact and therefore provides special support for strings to make their use convenient
- Every string is an instance of Java's built in `String` class, thus, Strings are objects.

Declaration

- Like any object, a string can be created using `new` as in the following example:

```
String str1 = new String("Hello dear");
```

- However, as an extra support, Java allows String object to be created without the use of new, as in:

```
String str2="How are you";
```

- This is inconsistent with the way Java treats other classes.

Substrings

- String objects are represented as a sequence of characters indexed from 0.

Example: **String greeting = "Hello, World";**

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

- A common operation on Strings is extracting a substring from a a given string.
- Java provides two methods for this operation:

substring(start);	Returns the substring from start to the end of the string
substring(start, end);	Returns a substring from start to end but not including the character at end.

Examples:

String sub = greeting.substring(0, 4); → "Hell"

String w = greeting.substring(7, 12); → "World"

String tail = greeting.substring(7); → "World!"

Concatenation Operator

- Another common operation on String is concatenation.
- As another special support for String, Java overloaded the + operator to be used to concatenate two String objects to get a bigger one.

```
String firstName = "Joseph";
String lastName = "Douglas";
String fullName = lastName+"  "+firstName;
```

→ "Douglas Joseph"

- If one of the operands of the + operator is a string, the other is automatically converted to string and the two strings are then concatenated.

```
String course = "BAIS";
int code = 102;
String courseCode = course+code
```

→ "BAIS102"

- We frequently use the concatenation operator in *println* statements.

```
System.out.println("The area ="+area);
```

- You need to be careful with concatenation operator. For example, what do you this the following print?

```
System.out.println("Sum ="+5+6);
```

Strings are Immutable

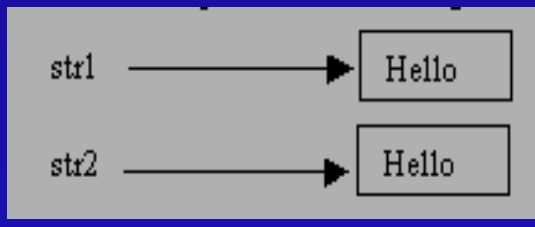
- Another special feature of Strings is that they are immutable. That is, once a string object is created, its content cannot be changed. For example, consider the following:

```
String str1 = "Hello World";
str1 = str1.substring(4);
```

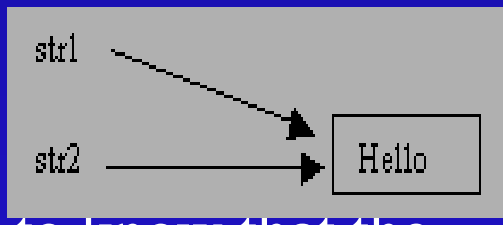
- Instead of changing the str1 object, another object is created. The former is garbage collected, thus, the reference to the former is lost.
- The fact that Strings are immutable allows the Java system to process Strings very efficiently.
- For example, consider the following:

```
String str1 = "Hello";
String str2 = "Hello";
```

- We would expect the following



- But in fact, this is what happens



- The Java system is smart enough to know that the two strings are identical and allocates same memory location for the two objects

Methods of the String Class

- **In addition to the substring methods, several predefined methods are provided in the built-in String class. Some of these are:**

int **length**()	returns the number of characters in this String
String **toUpperCase**() String **toLowerCase**()	returns a new String, equivalent to the Upper/lower case of this String
boolean **equals**(s) Boolean **equalsIgnoreCase**(s)	returns true if *s* the same length, as this String.
int **compareTo**(String s) int **compareToIngoreCase**(s)	returns +ve number, 0, or -ve number if this String is greater than, equal to or less than *s*.
char **charAt**(index)	returns the char in this String, at the *index* position.
int **indexOf**(ch) int **lastIndexOf**(ch)	Returns the index of the first / last occurrence of *ch* in this string, If not fount-1 is returned
String **trim**()	returns a String, obtained by removing spaces from the start and end of this string.
static String **valueOf** (any primitive type)	Returns a String representation of the argument
String **concat**(s)	equivalent to + symbol

Example using methods of String class

- The following program generates a password for a student using his initials and age.

```
public class MakePassword {
  public static void main(String[] args) {
      String firstName = "Amr";
      String middleName = "Samir";
      String lastName = "Al-Ibrahim";

      //extract initials
      String initials =
           firstName.substring(0,1)+
           middleName.substring(0,1)+
           lastName.substring(3,4);

      //append age
      int age = 20;

      String password =
  initials.toLowerCase()+age;

      System.out.println("Your Password
  ="+password);
  }
}
```

Formatting floating-point numbers

- Some times we would like to print floating point numbers only up to certain number of decimal places.
- For example we would want print the cost of an item in the form: SR16.50
- To achieve this, we use the getNumberInstance method of the the numberFormat class of the java.text package, to create an object.
- We then use the setMaximumFractionDigits and setMinimumFractionDigits methods to set the decimal places required.
- Finally, we use the format method to format our number.
- The format method returns a string, so we can print it.
- The example in the next slide demonstrate this process.

Formatting floating-point numbers

```
import java.text.NumberFormat;
public class NumberFormatting {
  public static void main(String[] args) {
      NumberFormat formatter =
  NumberFormat.getNumberInstance();
     formatter.setMaximumFractionDigits(2);
     formatter.setMinimumFractionDigits(2);

      double annualSalary = 72500;
      double monthlySalary =
  annualSalary/12;
      System.out.println("Unformatted
  Salary: SR"+monthlySalary);
      System.out.println("Formatted Salary:
  SR"+formatter.format(monthlySalary));
  }
}
```

```
D:\PROGRA~1\JCreator\GE2001.exe
Unformatted Salary: SR6041.666666666667
Formatted Salary: SR6,041.67
Press any key to continue...
```

Formatting floating-point numbers

- We can also use the **DecimalFormat** class of the same package to format floating-point numbers.
- In this approach, we create an object of the DecimalFormat class, specifying the format we want as a parameter.
- The difference between this approach and the previous is that the comma is not used by default.

```
import java.text.DecimalFormat;
public class DecimalFormatting {
    public static void main(String[] args) {
        DecimalFormat formatter = new
    DecimalFormat("0.00");
        double celsius = 37;
        double fahrenheit = 9.0/5.0*celsius+32;
        System.out.println("Unformatted: "+celsius+"oC =
    "+fahrenheit+"oF");
        System.out.println("Formatted: "+celsius+"oC =
    "+formatter.format(fahrenheit)+"oF");
    }
}
```

```
D:\PROGRA~1\JCreator\GE2001.exe
Unformatted: 37.0oC = 98.60000000000001oF
Formatted: 37.0oC = 98.60oF
Press any key to continue...
```