Contents

1. Introduction
2. Types and Variables
3. Statements and Control Flow
4. Reading Input
5. Classes and Objects
6. Arrays
7. Methods
8. Scope and Lifetime
9. Utility classes
10. Introduction to Object-Oriented Analysis and Design
Chapter 5: Classes and Objects

After this chapter you will be able to:

- Declare object variables
- Create objects
- Incorporate String objects
- Use different String operations
Object-Oriented Terms

- **Classes** specify what operations an object supports
- A *Car* class might define *start* and *stop*
- An **object** is an *instance* of some class
Another Example

- **Class Person** specifies the behavior of a person

  \[\text{class Person} \rightarrow \text{fred} \rightarrow \text{sally} \rightarrow \text{rachel} \rightarrow \text{Objects}\]

- Fred, Sally and Rachel are objects of type Person
Memory

- Memory is sequence of cells which can be identified using addresses or names (program variables)

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

```
int x = 4;
```

Copyright (c) 1999-2004 N. Afshartous
Declaring Object Variables

• Once a class is defined, the class name can be used as a type in a variable declaration
• A predefined String class exists in Java

```java
String x;
```

• x is a variable that can store the memory address of a String object
• We will only be instantiating existing classes in this course
Creating Objects

• The \texttt{new} operator creates objects

• Syntax: \texttt{new ClassConstructor(args)}

\begin{verbatim}
new String(“hello”);
\end{verbatim}

• The class constructor is a special operation provided by a class

• A class constructor \texttt{always} has the same name as its class
Initializing String Variables

• An object can be assigned to an object variable

\[
\text{String } x = \text{new String(“hello”);} \\
\]

• \(x\) stores the \textbf{address} of a String object

\[
\text{x} \quad \text{“hello”} \\
\]
String Init Shorthand

• Either the long or short form may be used

```java
String x = new String("hello");
x -> "hello"
```

```java
String y = "hello";
y -> "hello"
```
String Assignment

• A new string may be subsequently assigned to a string variable

```java
String x = "hello";
x = "there";
```

The object storing “hello” is no longer accessible and will be garbage collected.
String Assignment

• Assignment between object variables copies addresses

String x = "hello", y;

y = x;

• x and y store the same address
Assignment with Primitive Types

• Assignment involving variables having primitive types copies values

```
int x = 44, y = 33;
x = y;
```

Object variables store addresses, variables of primitive type store values
String CONCATENATION

• The plus operator (+) concatenates strings

String x = "hello";
String y = " there", z;
int a = 4;

z = x + y;

System.out.println("z = " + z);
System.out.println("a = " + a);

Output

z = hello there
a = 4

Output
Methods

• A method is a procedure defined in a class
• The String class defines the `charAt` method
• Signature: `char charAt (int)`

• Lookup `charAt` under class String in API doc
Using charAt

- **charAt** is used to extract individual characters from a String
- Indexing starts from 0

```java
String x = "hello";
System.out.println(x.charAt(0));
System.out.println(x.charAt(1));
```

**Output**

```
h
e
```
String Lengths

• Can find out the length of any String using \textit{Var.length()}

```java
String x = "hello";

for (int i = 0; i < x.length(); i++)
    System.out.println(x.charAt(i));
```

Output

```
  h
  e
  l
  l
  o
```
Printing A String In Reverse

String x = "Java Mania!";

for (int i = x.length() - 1; i >= 0; i--)
    System.out.print(x.charAt(i));
System.out.println();

Output !ainaM avaJ
Printing A String In Reverse (Another Way)

String x = "Java Mania!";
String y = "";

for (int i = x.length() - 1; i >= 0; i--)
    y += x.charAt(i);
System.out.println(y);

Output: !ainaM avaJ
Comparing Strings

- The equal operator (==) when comparing objects just compares addresses

```java
String x = "Java";
String y = x;
String z = "Java";

if (x == y)
    System.out.println("x == y");

if (x != z)
    System.out.println("x != z");
```

Output

```
x == y
x != z
```

Output

```
"Java"

x
y
z
```

Copyright (c) 1999-2004 N. Afshartous
Equals Method

• The String class has an equals method that compares the String values (not addresses)

```java
String x = "Java";
String y = x;
String z = "Java";

if (x.equals(z))
    System.out.println("x equals z");

if (x != z)
    System.out.println("x != z");
```

Output

```
x equals z
x != z
```
Upper and Lower Case Conversion

String x = "Java";
String y = " Rules";
String a = x.toUpperCase();
String b = y.toLowerCase();

System.out.println(x + y);
System.out.println(a + b);

• Output ?
String Methods

- public int compareTo(String anotherString)
- public String concat(String str)
- public String substring(int beginIndex)
- public String toLowerCase()
- public String toUpperCase()
- public boolean startsWith(String prefix)
It’s Exercise Time