SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on *Java actually: A Comprehensive Primer in Programming* (Chapter 4)

Topics Covered

- We use abstractions to handle the diversity that surrounds us in everyday life
- An abstraction represents the relevant properties of an object required to solve the problem at hand
- We need to represent the properties and behaviour of these abstractions
- In Java, abstractions can be represented by classes
- A class describes objects of a particular type
- It specifies the properties and behaviour of these objects

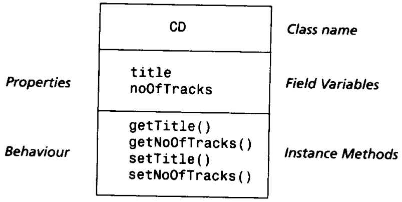
Introduction and Learning Objectives

Learning Objectives:

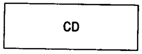
- The relationship between a class and its objects
- Representing the properties and behaviour of an object
- Creating objects using the new operator
- Manipulating objects by reference variables
- Calling methods on objects and accessing fields in objects
- Representing characters in the computer
- Using methods from the String class
- Reference equality versus value equality for objects
- Using primitive values as objects

Example

- A CD has certain properties:
  - a title
  - a number of tracks
- It should be possible to determine its title and how many tracks there are on it
- We should be able to change the title and number of tracks on the CD



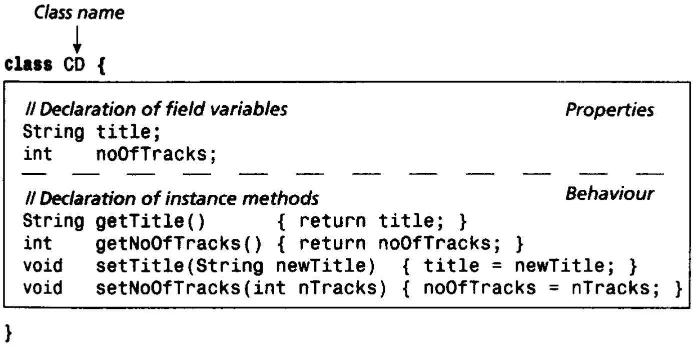
(a) Standard notation for a class



(b) Simplified notation for a class

Using Objects
Introduction to the object model
Abstractions, classes and objects

- A class declaration contains a number of declarations that define the properties and behaviour of its objects



```
class CD {
  // Declaration of field variables
  String title;
  int    noOfTracks;

  // Declaration of instance methods
  String getTitle()      { return title; }
  int    getNoOfTracks() { return noOfTracks; }
  void   setTitle(String newTitle)  { title = newTitle; }
  void   setNoOfTracks(int nTracks) { noOfTracks = nTracks; }
}
```

Reference variable

- A class is a "blueprint" for creating objects that have properties and behaviour defined by the class
- The term instance is often used as a synonym for an object
- There is only one CD class, but we can create several CD objects
- When we create an object from a class, we get **a reference value** for the newly-created object
- Each object of a class is unique

- The identity of the object is indicated by **the reference value**
- A **reference variable** (or **reference**) is a variable that can store a reference value of an object
- References are analogous to variables of primitive data types
- We manipulate an object via a reference that holds the reference value of the object

Reference variable declaration

- It is used to declare a reference variable
- It specifies the name of the reference and its reference type
- A class is a reference type
- References can only refer to objects of the specified class

$\boxed{\textbf{CD favouriteAlbum;}}$

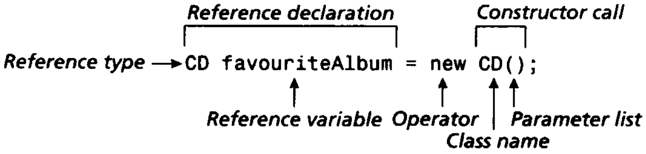
- A memory is allocated for the reference favouriteAlbum to store the reference value of a CD object

- No object is created as a result of declaring a reference
- To create an object of the class CD:

$\boxed{\textbf{new CD();}}$

- **new CD();** has two parts:
  - The operator **new**
  - A constructor call: **CD ( )** It specifies the name of the classt + specifies a list of parameter values
- The operator new
  - creates an object of class CD
  - returns the reference value of the new object
- The constructor call can be used to initialize the field variables in the new object

Often we combine the declaration of a reference and the creation of an object



```
CD favouriteAlbum = new CD();
```

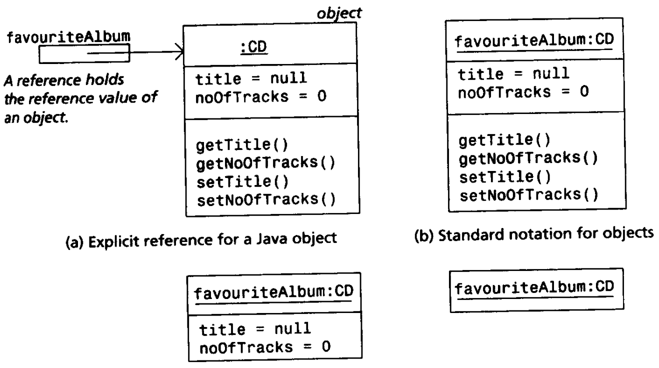
- The fields **title** and **noOfTracks** have the values **null** and **0** respectively

- After an object has been created, a reference that refers to the object can be used to send messages to the object
- Messages take the form of a method call in Java
- A method call to an object specifies:
  - the reference to the receiving object
  - the name of the method that is to be executed
  - any other information (in a parameter list)
- The class of the referred object must define the method that is called

Using Objects

Introduction to the object model

Using objects

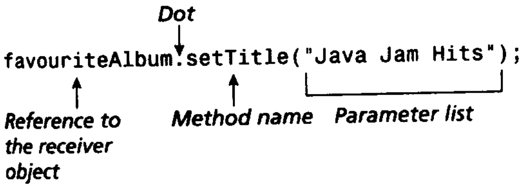


(a) Explicit reference for a Java object

(b) Standard notation for objects

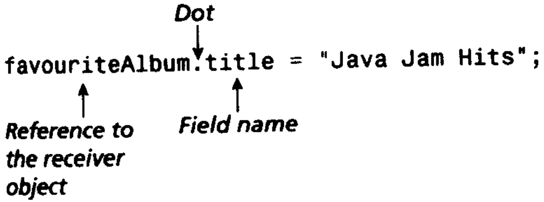
(c) Other simplified notations for objects

Using Objects
Introduction to the object model
Using objects



```
favouriteAlbum.setTitle("Java Jam Hits");
```

(a) Calling a method in an object

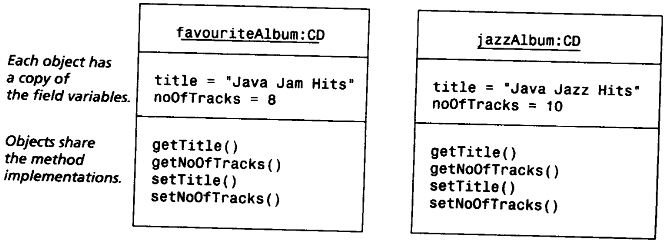


```
favouriteAlbum.title = "Java Jam Hits";
```

(b) Referring to a field in an object

- Each object has its own copy of field variables
- The fields of different objects of class CD can have different values
- The behaviour of an object is given by the instance methods
- The code that constitutes a method declaration is called a method implementation
- Objects of the same class share method implementations

Figure: Object state



Using Objects
Introduction to the object model
Object state

```java
// Using CD-objects
public class CDSampler {
  public static void main(String[] args) {
    // Create 2 CDs.
    CD favouriteAlbum = new CD();
    CD jazzAlbum = new CD();

    // Set state of the CDs.
    favouriteAlbum.setTitle("Java Jam Hits");
    favouriteAlbum.setNoOfTracks(8);
    jazzAlbum.setTitle("Java Jazz Hits");
    jazzAlbum.setNoOfTracks(10);

    // Print state of the CDs.
    System.out.println("Title of favourite album: " +
                       favouriteAlbum.getTitle());
    System.out.println("Number of tracks on favourite album: " +
                       favouriteAlbum.getNoOfTracks());
    System.out.println("Title of jazz album: " + jazzAlbum.getTitle());
    System.out.println("Number of tracks on jazz album: " +
                       jazzAlbum.getNoOfTracks());
  }
}
```

Program Output

```
Title of favourite album: Java Jam Hits
Number of tracks on favourite album: 8
Title of jazz album: Java Jazz Hits
Number of tracks on jazz album: 10
```

Characters and strings

- In programming languages, text is usually a sequence of characters and is called a text string, or just string
- Java provides a primitive data type **char**
- Java provides a pre-defined class **String**
- Each character is represented by an integer value called the **code number**
- Java uses a standard called Unicode to represent characters
- This standard assigns a unique code number for each character

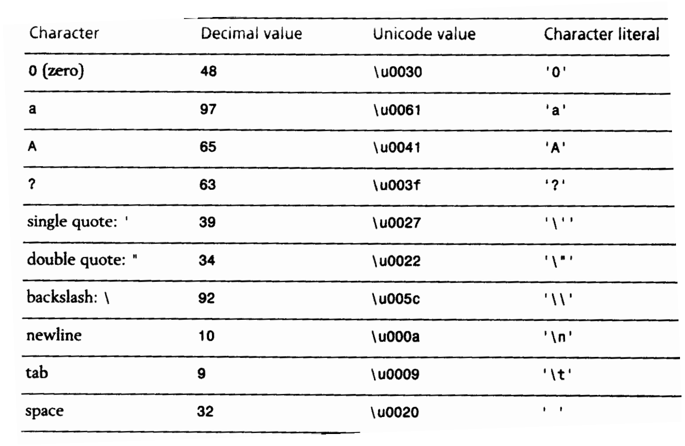
- The char data type represents the code number of each character as a 16-bit integer value
- We can represent 2^{16} characters in the data type
- We can represent the characters found in most of the languages in the world
- The Unicode values are usually specified as hexadecimal numbers

Example

- The letter **'a'** has the Unicode value **u0061**
- The digit **'0'** has the Unicode value **u0030**
- The character **'?'** has the Unicode value **u20ac**

- In Java, we can write a character as a char value
- The letter a can be written as **'a'** or **'\u0061'**
- Without the single quotes, the character a alone will be interpreted as a one-letter name
- The single quote ' by a backslash \ are needed
- The backslash character \ is used to "escape" the special meaning of a character

Using Objects
Strings
Character literals



Character	Decimal value	Unicode value	Character literal
0 (zero)	48	\u0030	'0'
a	97	\u0061	'a'
A	65	\u0041	'A'
?	63	\u003f	'?'
single quote: '	39	\u0027	'\''
double quote: "	34	\u0022	'\"'
backslash: \	92	\u005c	'\\'
newline	10	\u000a	'\n'
tab	9	\u0009	'\t'
space	32	\u0020	' '

Character var. and arith. expressions

- A character literal has the data type char
- We can declare variables that can store characters

  char newline = '\n', tab = '\u0009';
  char char1, char2, char3, char4;
  char1 = char4 = 'a'; char2 = char3 = 'b';

- A character can be an integer operand in an arithmetic expression (as it is represented by integer):

  int number = '5' - '0'; 53 - 48 gives 5

  int sumCodeNumbers = char1 + char2 + char3 + char4; 97+98+98+97 gives 390

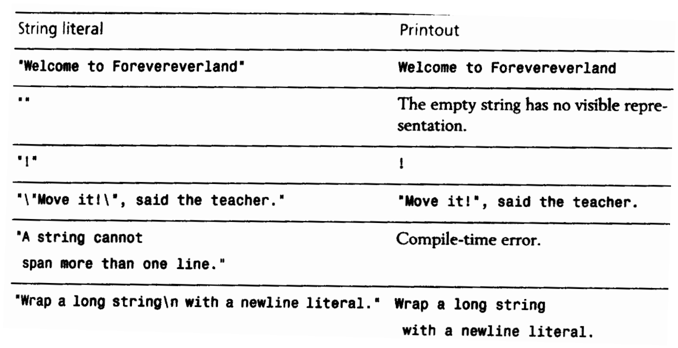
- $[a, \cdots, z]$, $[A, \cdots, Z]$, and $[0, \cdots, 9]$ are numbered consecutively in the Unicode standard
- We can compare characters and it is the code numbers that are actually compared:

  boolean test1 = (char1 == char4);

- Analogous to character literals, we can define string literals

- A string literals is defined by enclosing a sequence of characters in double quotes "
- String literals are objects of the class String
- The string literal "**abba**" is a String object
- This object stores the characters as a sequence
- Any double quotes " that actually occur in a string must be escaped with a backslash \
- String literals cannot span more than one line in the source code

Using Objects
Strings
String literals



String literal	Printout
"Welcome to Forevereverland"	Welcome to Forevereverland
""	The empty string has no visible representation.
"!"	!
"\"Move it!\", said the teacher."	"Move it!", said the teacher.
"A string cannot span more than one line."	Compile-time error.
"Wrap a long string\n with a newline literal."	Wrap a long string with a newline literal.

- We can declare variables of class String that can refer to string literals

  String firstName = "John", lastName = "Eriksen";

- The character sequence in a String object cannot be modified

- Seemingly modifying the string in a String object actually result in a new String object

- The binary operator + is used for concatenating two strings

  String fullName = firstName + "␣" + lastName;

String concatenation

```java
// Illustrating string concatenation
public class StringConcatenation {
  public static void main(String[] args) {
    String course = "programming";
    course = "Introductory course in " + course;                  // (1)
    System.out.println("course: " + course);
    int courseNumber = 100;
    String course1 = "C" + "S" + courseNumber + ": " + course; // (2)
    String course2 = 'C' + 'S' + courseNumber + ": " + course; // (3)
    System.out.println("course1: " + course1);
    System.out.println("course2: " + course2);
    System.out.println((int)'C');
    System.out.println((int)'S');
  }
}
```

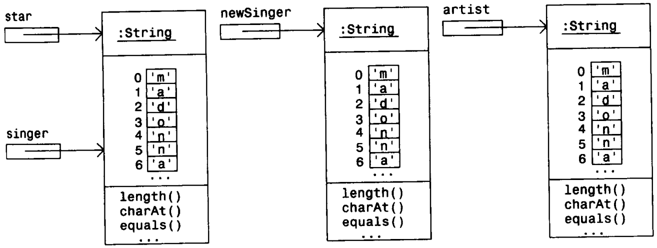
Program Output

```
course: Introductory course in programming
course1: CS100: Introductory course in programming
course2: 250: Introductory course in programming
67
83
```

Creating string objects

- Specification of a string literal in the program $\Longrightarrow$ creation of a String object
- The reference value of this object can be assigned to a String reference variable:
  String star = "madonna";
- If several reference variables are assigned the same string literal, they are aliases
  String singer = "madonna"; The reference singer is the same as star
- Another way of creating String objects is by using the **new** operator
  String newSinger = new String("madonna"); (1)
  String artist = new String(newSinger); (2)

Creating string objects



```
String star      = "madonna";
String singer    = "madonna";
String newSinger = new String("madonna");
String artist    = new String(newSinger);
```

Using Objects
Strings
String comparison

- Comparison of strings is based on lexicographical order
- The method **compareTo()** in the String class can be used to compare strings
- We call this method on one string and send the second string as a parameter in the method call

Example

```
int result1 = star.compareTo(singer);                    == 0
int result2 = star.compareTo(newSinger);                 == 0
String group1 = "abba", group2 = 'aha";
int result3 = group2.compareTo(group1);                   > 0
int result4 = group.compareTo(group2);                    <0
if (result4 ¡ 0) {
      System.out.println(group1 + " is smaller!");   Prints: abba is smaller!
}
```

Using Objects
Strings
Methods from the String class

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

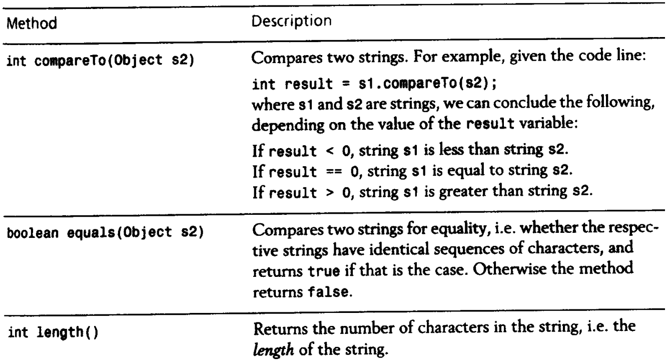
Intro. & Learning Objectives

Introduction to the object model

Strings
Characters and strings
Character literals
Character variables and arithmetic expressions
String literals
String concatenation
Creating string objects
String comparison
Methods from the String class

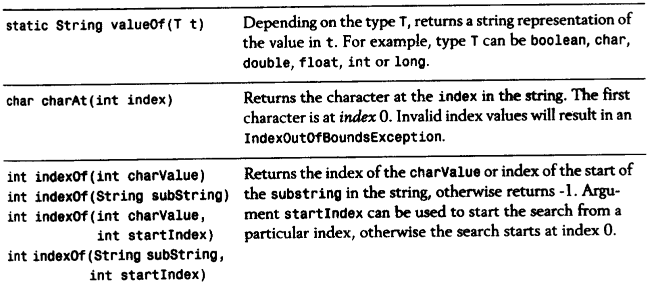
Manipulating references

Primitive values as objects



Method	Description
`int compareTo(Object s2)`	Compares two strings. For example, given the code line: `int result = s1.compareTo(s2);` where `s1` and `s2` are strings, we can conclude the following, depending on the value of the `result` variable: If `result < 0`, string `s1` is less than string `s2`. If `result == 0`, string `s1` is equal to string `s2`. If `result > 0`, string `s1` is greater than string `s2`.
`boolean equals(Object s2)`	Compares two strings for equality, i.e. whether the respective strings have identical sequences of characters, and returns `true` if that is the case. Otherwise the method returns `false`.
`int length()`	Returns the number of characters in the string, i.e. the *length* of the string.

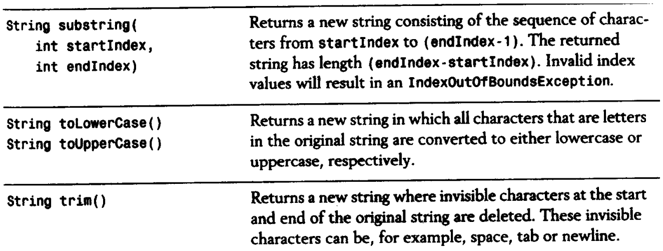
Using Objects
Strings
Methods from the String class



`static String valueOf(T t)`	Depending on the type `T`, returns a string representation of the value in `t`. For example, type `T` can be `boolean`, `char`, `double`, `float`, `int` or `long`.
`char charAt(int index)`	Returns the character at the `index` in the string. The first character is at *index* 0. Invalid index values will result in an `IndexOutOfBoundsException`.

`int indexOf(int charValue)`
`int indexOf(String subString)`
`int indexOf(int charValue, int startIndex)`
`int indexOf(String subString, int startIndex)`
	Returns the index of the `charValue` or index of the start of the `substring` in the string, otherwise returns -1. Argument `startIndex` can be used to start the search from a particular index, otherwise the search starts at index 0.

Using Objects
Strings
Methods from the String class



`String substring( int startIndex, int endIndex)`	Returns a new string consisting of the sequence of characters from `startIndex` to `(endIndex-1)`. The returned string has length `(endIndex-startIndex)`. Invalid index values will result in an `IndexOutOfBoundsException`.
`String toLowerCase()` `String toUpperCase()`	Returns a new string in which all characters that are letters in the original string are converted to either lowercase or uppercase, respectively.
`String trim()`	Returns a new string where invisible characters at the start and end of the original string are deleted. These invisible characters can be, for example, space, tab or newline.

```
// Illustrating misc. String methods
public class MiscStringMethods {
  public static void main(String[] args) {
    String group1 = "abba", group2 = "aha";
    int result3 = group2.compareTo(group1);            // > 0
//  int result4 = group2.compareTo(new Integer(10)); // (1) Error!
    if (result3 > 0) // True in this case.
      // "aha" is greater lexicographically.
      System.out.println(group2 + " is greater lexicographically!");
    if (group1.length() > group2.length())              // 4 > 3
      // "abba" is greater in length.
      System.out.println(group1 + " is greater in length!");

    String star = "madonna";
    int strLength = star.length();                     // 7
    System.out.println(star.charAt(strLength-4));      // o (index: 3,
                                                       //    i.e. 4th.
                                                             char)
    System.out.println(star.indexOf('n'));             // 4
    System.out.println(star.substring(0,3));           // "mad"
  }
}
```

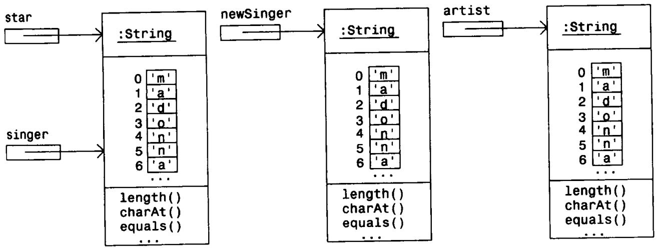
Program Output

```
aha is greater lexicographically!
abba is greater in length!
o
4
mad
```

- A reference value identifies an object in the computer's memory
- A Java object can only be referenced by its reference value
- A class defines a data type called a reference type
- A reference variable of a specific reference type can only store reference values of objects of that reference type
- We can change the reference value stored in a reference variable

- The same reference value can be assigned to several reference variables
- What happens when a reference value is assigned to several reference variables?
  - these variables are called aliases for the object identified by the reference value stored in them
  - an object can be manipulated by any of its aliases

Reference variables **star** and **singer** are aliases for the same String object



```
String star      = "madonna";
String singer    = "madonna";
String newSinger = new String("madonna");
String artist    = new String(newSinger);
```

The null literal

- The literal **null** is a special reference value
- It can be assigned to any reference variable
- **null** indicates that the reference variable does not refer to any object
- After assignment of null to a reference, the object previously referred to will no longer be available via this reference
- If we use a reference that has the value null, a runtime error (NullPointerException) can occur

Using Objects
Manipulating references
The null literal

```java
// Illustrating aliases
public class ReferenceValueSwapping {
  public static void main(String[] args) {
    String group1 = "abba", group2 = "aha", groupName;     // (1)
    groupName = group1;                                    // (2)
    group1 = group2;                                       // (3)
    group2 = groupName;                                    // (4)
    groupName = null;                                      // (5)
    System.out.println("group1 refers to: " + group1);
    System.out.println("group2 refers to: " + group2);
    System.out.println("groupName refers to: " + groupName);
    System.out.println(groupName.length());                // (6)
  }
}
```

Program Output

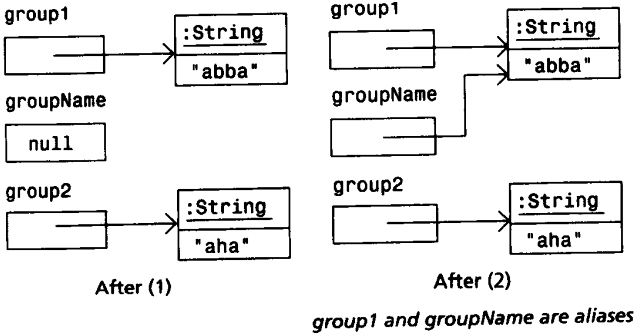
```
group1 refers to: aha
group2 refers to: abba
groupName refers to: null
Exception in thread "main" java.lang.NullPointerException
at ReferenceValueSwapping.main(ReferenceValueSwapping.java:12)
```

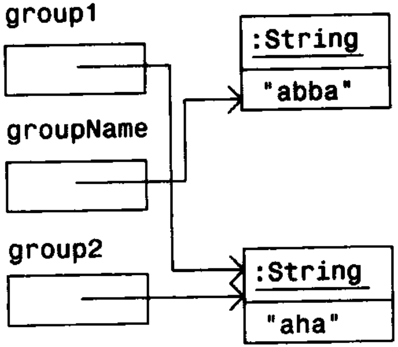
- In String comparison, we had value equality and reference equality
- We can now generalize these comparison to other objects
- What does it mean if we say that two cars are equal?
- To compare objects for value equality, the class must provide its own implementation of the equals() method

- This method has a special position in Java, and is used for comparing two objects for value equality
- The equals () method must check that it is meaningful to compare the two objects for value equality **(use <obj> instanceof <Class>)**
- The class String implements its own equals ()
- The operator == can be used to determine whether two references are aliases
- == compares the reference values stored in the references

Back to the previous program ...

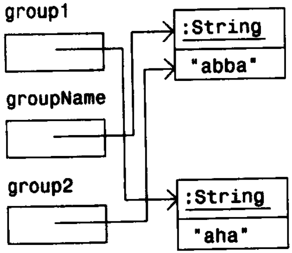
```
// Illustrating aliases
public class ReferenceValueSwapping {
  public static void main(String[] args) {
    String group1 = "abba", group2 = "aha", groupName;      // (1)
    groupName = group1;                                      // (2)
    group1 = group2;                                         // (3)
    group2 = groupName;                                      // (4)
    groupName = null;                                        // (5)
    System.out.println("group1 refers to: " + group1);
    System.out.println("group2 refers to: " + group2);
    System.out.println("groupName refers to: " + groupName);
    System.out.println(groupName.length());                  // (6)
  }
}
```


group1
:String
"abba"
groupName
null
group2
:String
"aha"
After (1)
group1
:String
"abba"
groupName
group2
:String
"aha"
After (2)
group1 and groupName are aliases

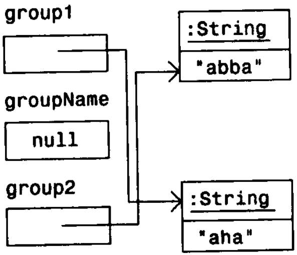


After (3)

group1 and group2 are aliases



After (4)



After (5)

groupName and group2 are aliases

Manipulating references

Comparing objects

```
public class Student {
    String name;
    String course;
  double average;
    public String getName ( )
    {
        return name;
    }
    public void setName (String studentName
         )
    {
        name = studentName;
    }
    public String getCourse ( )
    {
        return course;
    }
    public void setCourse (String
         studentCourse)
    {
        course = studentCourse;
    }
  public double getAverage ( )
    {
        return average;
    }
    public void setAverage (double
         studentAverage)
    {
        average = studentAverage;
    }
  /* equals */
  public boolean equals(Student stdt) {
    if (stdt instanceof Student) {
      if ((int)this.average == (int)stdt.
           average) return true;
    }
    return false;
  }
}
```

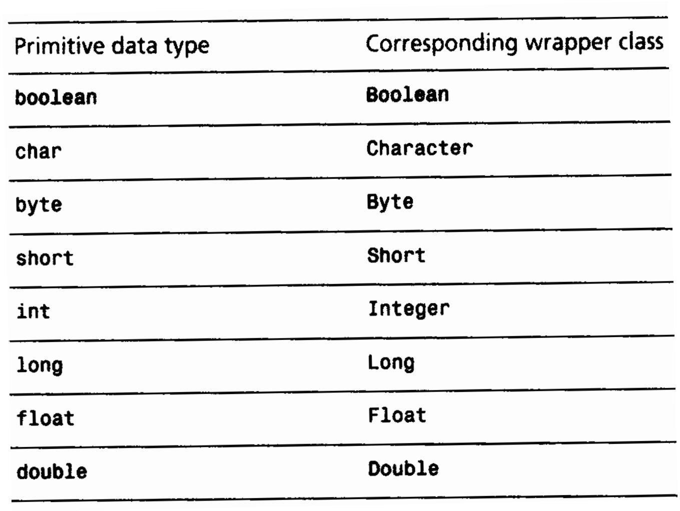
```
/*
 * SameAverage.java
 * To illustrate the usage of equals method
 */

public class SameAverage {
  public static void main(String[] args) {
    Student std1 = new Student(), std2 =
         new Student();

    std1.name = "John Do";
    std1.course = "2S03";
    std1.average = 75.5;

    std2.name = "Jane Smith";
    std2.course = "2S03";
    std2.average = 75.6876;

    System.out.printf("%n%nThe student %10s
          in the course %4s has an average
         of %3.2f.%n", std1.name, std1.
         course, std1.average);
    System.out.printf("The student %10s in
         the course %4s has an average of
         %3.2f.%n%n", std2.name, std2.
         course, std2.average);

    System.out.printf("It is %6s that the
         student %-10s and student %-10s
         have ALMOST the same average%n%n",
          std1.equals(std2), std1.name,
         std2.name);
  }
}
```

Primitive values as objects

- Primitive values are not objects
- Java offers **wrapper classes** so that values of primitive data types can be treated as objects
- The wrapper classes can be used to encapsulate primitive values
- There is a wrapper class for each primitive data type

Using Objects

Primitive values as objects



Primitive data type	Corresponding wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Auto-boxing

- Auto-boxing is the process of automatic conversion from a primitive value to a corresponding wrapper object

```
Integer iRef = 10;
```

- In the above example, the right-hand side of the assignment operator can be any int expression
- The value of the expression is evaluated and automatically encapsulated in an Integer object
- The reference value of the object is assigned to the reference variable iRef

Auto-unboxing

- Auto-unboxing is the process of automatic conversion from a wrapper object to the corresponding primitive value

```
int j = iRef;          // Auto-unboxing
```

- The right-hand side of the assignment operator can be any expression that evaluates to a reference value of an Integer object
- The int value encapsulated in the Integer object is assigned to the variable on the left-hand side

- We can also do **explicit conversion** between primitive data values and wrapper objects
- Wrapper classes have constructors that take a primitive value for encapsulation
- Wrapper classes have methods to read the value in the wrapper object

```
Integer iRef = new Integer(10);   //Explicit boxing
int j = iRef.intValue():  //Explicit unboxing
```

- The method **intValue()** in the class Integer returns the value in the wrapper object as an int value

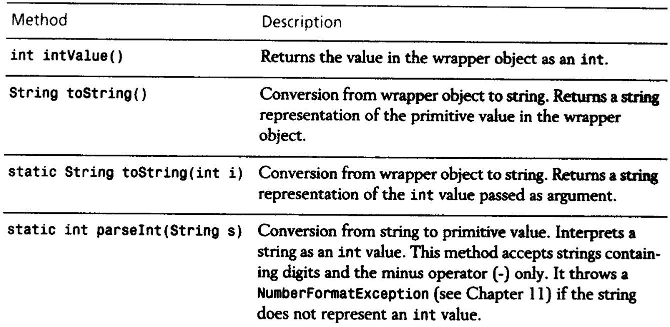
Using Objects
Primitive values as objects
Explicit boxing and unboxing

```
// Conversions: wrapper <—> primitive value
public class PrimitiveValueWrapper {
  public static void main(String[] args) {
    // A primitive value.
    int valueIn = 2006;

    // Two ways of creating an object from a primitive value:
    Integer valueObject;
    valueObject = new Integer(valueIn);
    valueObject = valueIn;            // Simple variant

    // Two ways of creating a primitive value from an object:
    int valueOut;
    valueOut = valueObject.intValue();
    valueOut = valueObject;           // Simple variant
    assert(valueIn == valueOut);      // Assert: same primitive value
    System.out.println("valueIn: " + valueIn + ", valueOut: " + valueOut
         );
  }
}
```

Program Output

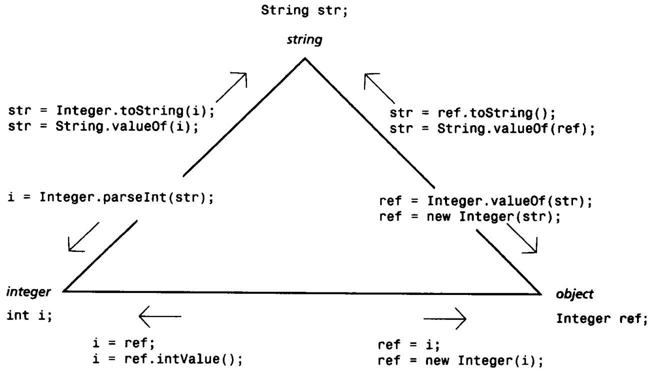
```
valueIn: 2006, valueOut: 2006
```

Using Objects
Primitive values as objects
Useful methods in the wrapper classes



Method	Description
`int intValue()`	Returns the value in the wrapper object as an `int`.
`String toString()`	Conversion from wrapper object to string. Returns a string representation of the primitive value in the wrapper object.
`static String toString(int i)`	Conversion from wrapper object to string. Returns a string representation of the `int` value passed as argument.
`static int parseInt(String s)`	Conversion from string to primitive value. Interprets a string as an `int` value. This method accepts strings containing digits and the minus operator (-) only. It throws a `NumberFormatException` (see Chapter 11) if the string does not represent an `int` value.

Using Objects
Primitive values as objects
Useful methods in the wrapper classes



```
// Conversions: string —> wrapper —> primitive
public class PrimitiveValueRepresentation {
  public static void main(String[] args) {
    String string1, string2;

    // Case A: string —> wrapper —> primitive —> string
    string1 = "2005";
    Integer iWrapper = new Integer(string1);
    int iPrimitive = iWrapper;
    string2 = Integer.toString(iPrimitive);
    assert(string1.equals(string2));  // (1)

    // Case B: string —> primitive —> wrapper —> string
    string1 = "12.5";
    double dPrimitive = Double.parseDouble(string1);
    Double dWrapper = dPrimitive;
    string2 = dWrapper.toString();
    assert(string1.equals(string2));  // (2)
  }
}
```