

Java characters

Lecture 8

Waterford Institute of Technology

January 31, 2016

John Fitzgerald

Presentation outline

Estimated duration presentation

Questions at end presentation

Topics discussed:

- Method and constructor parameters
- Boxing & Autoboxing
- String class
- Character class
- Randomness
- Conditional operator (?:)
- Precedence

Primitives

Passed as parameters

Primitive object passed as parameter:

- Copy passed to method
- Change copy in method
- Variable outside method unchanged

```
public class ArrayParameters {  
    public static void testPrimitiveParam () {  
        int val = 0;  
        System.out.println(val); //prints 0  
        modifyPrimitive(val);  
        System.out.println(val); //prints 0  
    }  
    public static void modifyPrimitive(int v) {  
        v = 100;  
    }  
}
```

Arrays

Passed as parameters

Array reference passed as parameter:

- Argument references same object before and after call
- Changes to array in method persist outside method

```
public class ArrayParameterPassing {  
    public static void testArrayParam() {  
        int[] ar = {1,2,3};  
        System.out.println(ar[2]);//prints 3  
        modifyArray(ar);  
        System.out.println(ar[2]);//prints 100  
    }  
    public static void modifyArray(int[] a) {  
        a[2] = 100;  
    }  
}
```

Wrapper class

AutoBoxing (Boxing & Unboxing)

- Applies to wrapper classes such as Integer, Byte and so on
- Eight wrapper classes Java 7
- Example: automatic conversion from primitive *int* to *Integer*
- Known as **boxing**
- Useful as keys in collections, e.g. ArrayList where primitives disallowed

```
//Boxing
//ArrayList<int> values: <-----not allowed
// Below i is autoboxed through method invocation.
ArrayList<Integer> values = new ArrayList<>();
for (int i = 0; i < 10; i += 1) {
    values.add(i);
}
```

Wrapper class

Unboxing

- Example: automatic conversion from *Integer* object to *int*
- Referred to as **unboxing**

```
ArrayList<Integer> values = new ArrayList<>();  
for (int i = 0; i < 10; i += 1) {  
    values.add(i);  
}  
int[] ar = new int[values.size()];  
for (int i = 0; i < ar.length; i += 1)  
{  
    ar[i] = values.get(i);  
    System.out.println(ar[i]);  
}
```

Wrapper class

Unboxing

```
ArrayList<Double> ar = new ArrayList<>();  
ar.add(3.1416); // PI is autoboxed through method invocation.  
//Unboxing through assignment  
double pi = ar.get(0);  
System.out.println("pi = " + pi);
```

Java String class

Has several constructors and methods

- `compareTo(String anotherString)`
- `toUpperCase()`
- `equalsIgnoreCase(String anotherString)`
- `length()`
- `valueOf(int i) //int to String`

```
//string literal
String str = new String("textwidth");
int length = str.length();
//converts textwidth to TEXTWIDTH
str.toUpperCase();
```

```
//creates string "100" referenced by str
String str = String.valueOf(100);
String str2 = "online";
//compares str and str2, returning false
//unless both strings equivalent
str.equalsIgnoreCase(str2);
```


Java String class

Manipulate characters in String object

- String an ordered collection characters
- Zero-indexed array
- `charAt(int pos)` accesses character at position pos

Concatenate string objects

- `concat(String str)` concatenates str to existing string
- Overloaded `+` operator achieves same effect

```
String str = "TEXT"  
str.charAt(0) is "T"  
str.charAt(1) is "E"  
str.charAt(2) is "X"  
str.charAt(3) is "T"
```

```
String s3 = "why ";  
String s4 = "not?";  
String s5 = s3 + s4;  
System.out.println("s3 + s4 " + s5);  
System.out.println("s3 + s4 "+s3.concat(s4));  
//both statements output: s3 + s4 why not?
```

String objects immutable

Health warning: String objects once created cannot be modified
References below list other similar cases such as wrapper classes, example:

- Integer
- Byte
- Long

```
String str = new String("java");  
//str is reference pointing to string object "java"  
str.toUpperCase();//has no effect. Produces no warning.  
System.out.println(str);//outputs java, not JAVA  
str = "JAVA";//reference str points to different object  
System.out.println(str);//outputs JAVA
```

String objects immutable

Easy to think otherwise

```
String s1 = new String("java");  
System.out.println(s1); // Output: java  
s1.toUpperCase();  
System.out.println(s1); // Output: java  
System.out.println(s1.toUpperCase()); // Output: JAVA
```

String

String literal v String object

State and explain the outputs from following snippets.

```
// String literals (interned)
String a = "abcd";
String b = "abcd";
System.out.println("a equals b : " + (a == b)); // true
System.out.println("c equals b : " + b.equals(c)); // true
```

```
// String objects
String c = new String("abcd");
String d = new String("abcd");
System.out.println("c equals d : " + (c == d)); // false
System.out.println("c equals d : " + c.equals(d)); // true
```

Character

Java primitive

- `char` is a primitive Java type
 - `char ch = 'a';`
 - `System.out.println(ch);`
 - outputs **a**
- Expose underlying integer representation
 - `int chInt = (int)ch;`
 - `System.out.println(chInt);`
 - outputs **97**

Character

Java wrapper class

- Facilitates use of *char* where object required

```
/* This code snippet outputs:  
 * a A  
 */  
Character c = new Character('a');  
ArrayList<Character> characters = new ArrayList<>();  
characters.add(c);  
characters.add('A');  
for (Character character : characters)  
{  
    System.out.print(character + " ");  
}
```

Character class

Some useful methods

- Determines if character `ch` is a digit
 - `static boolean isDigit(char ch)`
- Determines if character `ch` is a letter
 - `static boolean isLetter(char ch)`
- Determines if character `ch` is letter or digit
 - `static boolean isLetterOrDigit(char ch)`
- Determines if character `ch` is a lowercase
 - `static boolean isLowerCase(char ch)`
- Determines if character `ch` is upper case
 - `static boolean isUpperCase(char ch)`
- Determines if character `ch` is whitespace (space or tab)
 - `static boolean isWhitespace(char ch)`
- Converts character `ch` to lower case
 - `static char toLowerCase(char ch)`
- Converts character `ch` to upper case
 - `static char toUpperCase(char ch)`

Character class

Example isDigit method

```
public static void testIsDigit() {  
  
    Character ch = 'a';  
    boolean b = Character.isDigit(ch);  
  
    if (b) {  
        System.out.println("The character " + ch + " is a digit");  
    }  
    else {  
        System.out.println("The character " + ch + " is not a digit");  
    }  
}
```


Java primitive *char*

Arithmetic

- Because **char** has underlying integer representation
- May be used in arithmetic expressions
 - Example: 'A' convertible to 65
 - Example: 'B' convertible to 66
- Character arithmetic used in method `isValid`

```
static boolean isValid2(String pin)
{
    for (int i = 1; i < pin.length(); i++)
    {
        if ((pin.charAt(i) - pin.charAt(i-1)) != 1)
        {
            return true;
        }
    }
    return false;
}
```

Number systems

Used in computing

Binary (base 2)

- 0,1
- 26 in binary: `int binVal = 0b11010;`

Decimal (base 10)

- 0,1,2,3,4,5,6,7,8,9
- 26 in decimal: `int decVal = 26;`

Hexadecimal (base 16)

- 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- 26 in hex: `int hexVal = 0x1a`

Octal (base 8) sometimes used.

Binary	Decimal	Hexadecimal
1111	15	F
11111111	255	FF

Character encoding

Americal Standard Code for Information Exchange (ASCII)

Represents text in devices such as computers & printers

Character encoding scheme based on English alphabet

Encodes 128 specified characters, 33 non-printable, 95 printable

- 0 to 9
- a to z
- A to Z
- Some punctuation
- Control codes: example non-printable line feed, carriage return
- Blank space

Character encoding

Unicode standard

A superset of ASCII

Presently represents more than 110,000 characters

In excess of 100 scripts, for example:

- Left to right: Latin (English) Cyrillic (Russian) scripts
- Right to left: Arabic, Hebrew scripts
- Various symbols
- Implemented in:
 - Many operating systems
 - XML
 - Java

Several Unicode encodings exist

In excess 80% *www* uses UTF-8

- 1 byte (8 bits) used for ASCII chars

Character encoding

Examples

```
char c1 = 0x41;  
char c2 = 65;  
//Both output the letter A  
System.out.println("char1 : " + char1);  
System.out.println("char2 : " + char2);
```

ASCII Hex Symbol		
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g

ASCII Hex Symbol		
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w

Randomness

Generating random numbers

Many libraries available to generate (pseudo) random numbers

```
//Using Math.random()
//Returns a double value with a positive sign,
//greater than or equal to 0.0 and less than 1.0 : range [0 1).
StdOut.print("Pseudo-random number range [2,8] using Math library: ");
double rval = Math.random();
int rval1 = (int)(rval*7 + 2);
StdOut.print(rval1);
//Typical output: Pseudo-random number range [2,8] using Math library: 5
```

```
//Using java.util.Random
//Random.nextInt(int n) generates random number in range [0 n)
StdOut.print("\nPseudo-random number range [2,8] using java.util library: ");
Random random = new Random();
int rval2 = random.nextInt(7) + 2;
StdOut.print(rval2);
//Typical output: Pseudo-random number range [2,8] using java.util library: 3
```

Java primitive *char*

Arithmetic

- Generate a random character

```
public static char randomCharacter()
{
    return (char) ('A' + (int) (Math.random()*26));
}
```

- Test range

```
public static boolean isDigit(char ch)
{
    return (ch >= '0' && ch <= '9');
}
```

Unicode Special Characters

Table of Escape sequences

An escape sequence comprises a character preceded by backslash.

<code>\n</code>	Newline (moves to the next line)
<code>\b</code>	Backspace
<code>\f</code>	Form feed (starts a new page)
<code>\r</code>	Return to the beginning of the current line
<code>\t</code>	Tab (moves horizontally to the next tab stop)
<code>\\</code>	The backslash character itself
<code>\'</code>	The character ' (required only in character constants)
<code>\"</code>	The character " (required only in string constants)
<code>\ddd</code>	Character whose Unicode value octal number ddd

Java Operator

Ternary

Conditional operator ?:

- Also known as *ternary* operator
- Can be thought of as *if-then-else* operator
- If condition true assign value1 else value2

```
int value1 = 1;
int value2 = 2;
int result;
boolean someCondition = true;
result = someCondition ? value1 : value2;
```

Java Operator

Ternary

A method to return the absolute value of an integer:

```
public static int absoluteValue(int a)
{
    if (a < 0) {
        return -a;
    }
    return a;
}
```

Alternative versions using ternary or conditional operator

```
public static int absoluteValue(int a)
{
    return a < 0 ? -a : a;
}
```

Java Operator

Ternary

```
public static int absoluteValue(int a)
{
    return a < 0 ? -a : a;
}
```

If this condition is true

return this value

otherwise

return this value

Operator Precedence

See the complete table listed in references

Order of evaluation rules

- Highest precedence include parentheses and array access
- Multiplication & division before addition & subtraction
- Logical operators lower than multiplication
- Lowest precedences ternary followed by assignment
- If in doubt use parens

Operators	Precedence
postfix	<i>expr</i> ++ <i>expr</i> --
unary	++ <i>expr</i> -- <i>expr</i> !
multiplicative	* / %
additive	+ -
relational	< > <= >=
equality	!= ==
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %=

Summary

- String class
 - comprises character array
 - String objects immutable
- Character class
 - primitive type
 - can represent keyboard characters
 - has underlying integer representation
 - examined some of its methods
- Number systems
 - decimal
 - binary
 - hexadecimal
 - octal
- Character encoding
 - ASCII
 - Unicode, example UTF-8

Summary

- String class
 - One of most widely used classes
- Randomness
 - Math and java.util packages
 - how to generate random character
 - Unicode special characters
- Conditional operator (?:)
 - also known as *ternary* operator
- Precedence
 - rules for order of evaluation
- Arrays
 - zero index based
 - comparison with ArrayList
- Boxing & Autoboxing

Referenced Material

1. Operator Precedence

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>

[Accessed 2014-05-17]

2. Characters

<http://docs.oracle.com/javase/tutorial/java/data/characters.html>

[Accessed 2014-05-17]

3. ASCII

<http://en.wikipedia.org/wiki/ASCII>

[Accessed 2015-02-19]

Referenced Material

4. Unicode

`http://en.wikipedia.org/wiki/UTF-8`

`[Accessed 2015-02-19]`

5. Binary to Decimal to Hexadecimal Converter

`http://www.mathsisfun.com/`

`binary-decimal-hexadecimal-converter.html`

`[Accessed 2015-02-19]`

Referenced Material

6. ASCII Table

<http://ascii.cl/>

[Accessed 2015-02-19]

7. Immutable classes

<http://stackoverflow.com/questions/5124012/examples-of-immutable-classes> [Accessed 2015-02-23]