Java characters

Waterford Institute of Technology

June 16, 2014

John Fitzgerald

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)`

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`

```java
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;
}
```

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.

`\n`	Newline (moves to the next line)
`\b`	Backspace
`\f`	Form feed (starts a new page)
`\r`	Return to the beginning of the current line
`\t`	Tab (moves horizontally to the next tab stop)
`\\`	The backslash character itself
`\'`	The character ' (required only in character constants)
`\"`	The character " (required only in string constants)
`\ddd`	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 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

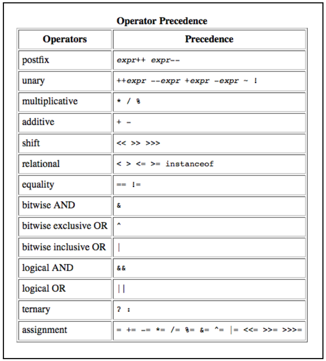
```
public static int absoluteValue(int a)
{
    return a < 0 ? -a : a;
}
```

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



Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Execution of class

main method

The *main* method starts class execution

- Hidden by default in BlueJ
- Possible to introduce explicitly
- Class not obligated to possess *main*
- Project of many classes: one *main* may be sufficient

```
public class TestShapes
{
    public static void main(String[] args)
    {
        Triangle triangle = new Triangle();
        triangle.draw();
    }
}
```

Execution of class

main method

The *main* method starts class execution

- Hidden by default in BlueJ
- More usually *main* introduced explicitly
- Class not obligated to possess *main*
- Project of many classes: one *main* sufficient

```java
public class TestShapes
{
    public static void main(String[] args)
    {
        Triangle triangle = new Triangle();
        triangle.draw();
    }
}
```

Execution of class

main method

Every Java application must possess *main* method

- public static
  - This order the convention
  - But may be any order
- Array of strings `String [ ] args`
- Facilitates input to program from runtime system
  - *args* name the convention
  - But not mandatory
  - Sometimes *argv*

class *Cross*:

- *args* resolves to integer
- No safety checks conducted

```
//signature of main method
public static void main(String[] args)
```

```
public class Cross {
  public static void main(String[] args)
  {
    int n = Integer.parseInt(args[0]);
     printCross(n);
  }
  public static void printCross(n)
  {
       ...
  }
}
```

Arrays

Passed as parameters

Array reference passed as parameter:

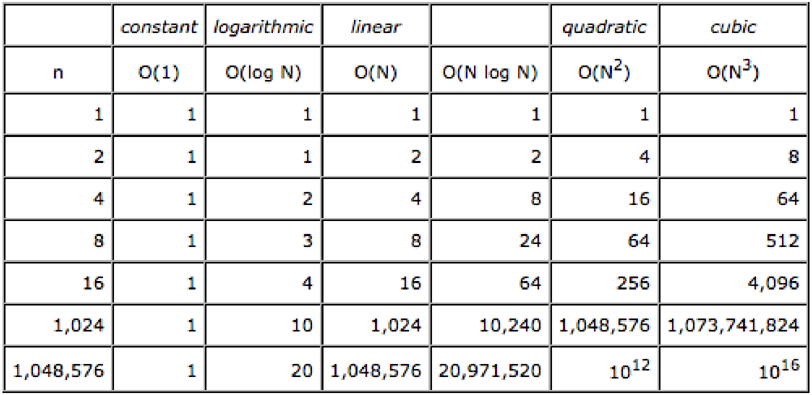
- Argument references same object before and after call
- Changes to array in method persist outside method

```
import sedgewick.stdlib.*;
public class ArrayParameters {
    public static void main(String[] args) {
        int[] ar = {1,2,3};
        StdArrayIO.print(ar);//prints 1 2 3
        modifyArray(ar);
        StdArrayIO.print(ar);//prints 1 2 100
    }
    public static void modifyArray(int[] a) {
        a[2] = 100;
    }
}
```

Big O Notation

Time classification of algorithms

- One method of estimating algorithmic processing time
- Benchmarking, an alternative, generally more accurate

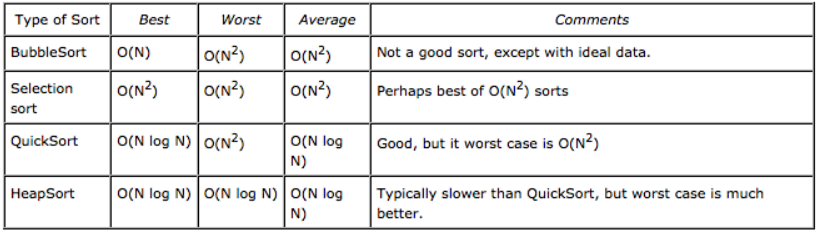


	constant	*logarithmic*	*linear*		*quadratic*	*cubic*
n	O(1)	O(log N)	O(N)	O(N log N)	O(N^2)	O(N^3)
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4,096
1,024	1	10	1,024	10,240	1,048,576	1,073,741,824
1,048,576	1	20	1,048,576	20,971,520	10^{12}	10^{16}

Big O Notation
Sorting

Important to have regard to

- Best
- Average
- Worst



Type of Sort	*Best*	*Worst*	*Average*	*Comments*
BubbleSort	O(N)	O(N^2)	O(N^2)	Not a good sort, except with ideal data.
Selection sort	O(N^2)	O(N^2)	O(N^2)	Perhaps best of O(N^2) sorts
QuickSort	O(N log N)	O(N^2)	O(N log N)	Good, but it worst case is O(N^2)
HeapSort	O(N log N)	O(N log N)	O(N log N)	Typically slower than QuickSort, but worst case is much better.

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
```

Enum

Special data type

Variable selectable from set predefined constants

- `enum Day {WEEKDAY, WEEKEND}`

```
enum Day {WEEKDAY, WEEKEND}
public class EnumTest {
    public static void makePlans(Day day) {
        switch (day) {
        case WEEKDAY:
            System.out.println("Working like a dog;");
            break;
        case WEEKEND:
            System.out.println("Sleeping like a log");
            break;
        default:
        }
    }
    public static void main(String[] args) {
        makePlans(Day.WEEKDAY);
    }
}
```

Referenced Material

1. Operator Precedence

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

[Accessed 2014-05-17]

2. Big O Notation

`http://www.leepoint.net/notes-java/algorithms/big-oh/bigoh.html`

[Accessed 2014-05-17]

Referenced Material

3. Characters

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

[Accessed 2014-05-17]

4. Enum Type

http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

[Accessed 2014-05-18]