CSC 3215: Object Oriented Programming - 1 (JAVA)

Statements, Expressions, Operators, and Control Flow

Dr. Kamruddin Nur

Associate Professor
Dept. of Computer Science, AIUB
kamruddin@aiub.edu

February, 2018

Contents

Statements

Statements:

- are like **sentences** in natural languages
- forms a complete **unit of execution**
- usually ends with a semicolon (;)
- Statements in Java can be one of the followings -
  - Variable **declaration, initialization or assignment**

```
1
2  // declaration statement
3  int a;
4
5  //assignment statement
6  a = 10;
7
8  //declaration and initialization
9  float b = 15.0f;
```

Statements *(Cont'd)*

- **Control Flow Statements**
  - Statements are generally executed from top to bottom
  - Control flow statements can be used to break the flow of execution such as -
  - **Decision making** statements (`if..else if..else, switch`)
  - Boolean expressions are those expressions which return either `true` or `false`
  - Boolean expressions use the relational (`>, >=, <, <=`)and equality operators (`!=, ==`) as well as logical AND (`&&`) and OR (`||`)
  - Boolean expressions can be used to control program flow

Statements *(Cont'd)*

```
1
2  if(boolean-expression1) {
3  // statements
4  }
5  else if(boolean-expression2) {
6  // statements
7  }
8  else
9  {
10 // statements
11 }
```

Statements *(Cont'd)*

```
1
2  switch(expression) {
3  case value1:
4  // statements
5  break;
6  case value2:
7  // statements
8  break;
9  default:
10 // statements
11 }
```

Statements *(Cont'd)*

– **Looping**(`for, while, do-while`)

```
1
2  while (boolean-expression){
3          //statements to be executed in a loop
4  }
5
6  do{
7  //statements to be executed first and then in a loop
8  }while (boolean-expression);
9
10
11 for(intial-expression; boolean-expression;
       increment-or-decrement){
12         //statements to be executed in a loop
13 }
14
15 int c = 1; //Iinitialization
16 while (c<10){
17 //statements
18 c++; //increment/decrement
19 }
```

Statements *(Cont'd)*

– **Branching** statements (`break, continue, return`) to conditionally execute particular blocks of code

- The `break` statement terminates the loop immediately, and the control of the program moves to the next statement following the loop.
- The `continue` statement skips the current iteration of a loop (for, while, and do...while loop).
- `return` returns with or without a value

Statements *(Cont'd)*

- **Expressions**
  - An **expression** is a series of variables, operators, and method calls that **evaluates** to a **single value**
  - Combination of **operators** and **operands** which evaluates to a single value
  - Segments of code (i.e. method call) that perform computations and return values
  - In Java, there are four kinds of expression statements:
  - **Assignment expressions**
  - **Prefix and postfix increment and decrement**
  - **Method invocations**, whether or not (void) they return a value
  - **Object creation expressions**, such as new Account

Statements *(Cont'd)*

```
1
2  // Arithmatic expression
3  int z = x * y;
4  a += 10;
5  //Increment / decrement
6  count++;
7  --count;
8
9  //Object creation expression
10 Scanner sc = new Scanner(System.in);
11 Account a = new Account();
12
13 //method call expression
14 String line = sc.nextLine();
15 a.showInfo();
```

Statements *(Cont'd)*

- **Blocks**
  - A block is a group of zero or more statements between balanced braces

```
1
2  if (condition == true) { // begin block 1
3
4          System.out.println("Condition is true."); //statement 1
5          return; //statement 2
6
7  } // end block one
```

Statements *(Cont'd)*

- **Comments**
  - Java supported comments

```
1
2  //single line comment
3
4  /* multi-line
5  comment */
6
7  /**
8  * Javadoc style of writing comment
9  */
```

2 Operators

Operator Precedence

Java Operators

- Arithmetic $(+, -, *, /, \%)$
- Assignment $(=, + =, - =, * =, / =)$
- Increment and decrement $(++, --)$
- Relational operators $(==, ! =, <, <=, >, >=)$
- Logical operators $(||, \&\&)$ (note: logical or, logical and)

Important: The order of operations is defined through **operator precedence**.

Java Operator Precedence

- **Operator precedence** **defines** how an expression evaluates when several operators are present / **determines** the **order** of evaluation

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Figure 1: Java Operator Precedence

Java Operator Precedence *(Cont'd)*

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

3 Java Reserved Keywords and Literals

Java Reserved Keywords and Literals

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Reserved Literals in Java				
null	true	false		

Figure 2: Java **Reserved Keywords and Literals**

Thanks

Thanks for your time and attention!

kamruddin@aiub.edu