Computer Programming

Simple and Compound Statements

Willy Picard
Department of Information Technology
The Poznan University of Economics
<picard@kti.ae.poznan.pl>

Agenda

- Lecture Goal(s)
- Variables and Scope
- Simple Statements
- Conditional Statements
- Loop Statements
- Branching Statements
- Conclusions

Lecture Goal(s)

Lectures Overview

Fundamental Concepts

- 1: Introduction
- 2: Basic data structures & $\boxed{\text{Statements}}$
- 3: Object-oriented programming I
- 4: Object-oriented programming II
- 5: Object-oriented programming III
- 6: Complex data structures
- 7: Threads & Exception handling

Today's Goal

- To provide programming knowledge about statements

Variable and Scope

Variable Definition

- A variable is an item of data named by an identifier

Java Identifiers

- Unlimited-length sequence of
  - Java letters — *A-Z, a-z, _, $*
  - Java digits — *0-9*
- First a letter
- Caution!
  - keywords
  - literals — *true* or *false*
  - null — *null*

Java Keywords

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

Java Identifier Examples

- Correct
  - `myVariable`
  - `my2ndVariable` (better: `mySecondVariable`)
  - `_internalVariable`
  - `_i_love_underscores`
  - `$legacyVariable`
- Incorrect
  - `1stVariable`
  - `vice-versa`

Variable Definition in Java

- With initialization
  - *type name = value;*

    ```
    int counter = 0;
    ```
- With late initialization
  - *type name;*
  - *name = value;*

    ```
    int counter;
    counter = 0;
    ```

Variable Classification in Java

- Four categories
  - member variable
  - local variable
  - method parameter
  - exception-handler parameter

```
class MyClass {
     ...
     member variable declaration
     ...
     public void myMethod(method parameters){
          ...
          local variable declaration
          ...
          catch (exception handler parameters) {
               ...
          }
          ...
     }
     ...
}
```

Example

Member variable

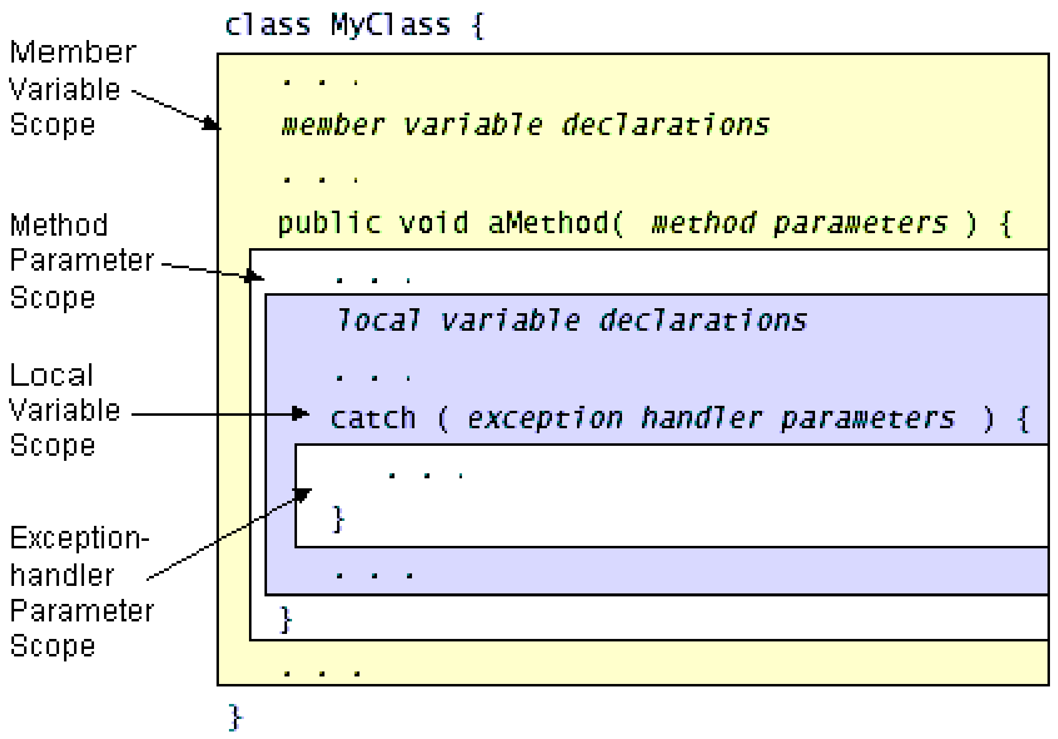
Local variable

Method parameter

```
package pl.poznan.ae.compProg;

import java.util.*;

public class Sorter {
 private List  words;

  public void sort(String[] words){
    _words = Arrays.asList(words);
    Collections.sort(_words);
  }
  public String getSortedWords(){
    String sortedString = "";
    for (int i = 0; i< _words.size(); i++){
      sortedString += _words.get(i);
    }
    return sortedString;
  }
 public static void main(String[] args){
    Sorter sorter = new Sorter();
    sorter.sort(args);
    System.out.println(sorter.getSortedWords());
 }
}
```

Scope

- Where a variable can be referred
- Scope ≠ visibility



```
class MyClass {
    . . .
    member variable declarations
    . . .
    public void aMethod( method parameters ) {
        . . .
        local variable declarations
        . . .
        catch ( exception handler parameters ) {
            . . .
        }
        . . .
    }
    . . .
}
```

Simple Statements

Statement Definition

- A statement is a complete unit of execution

Empty Statement

- Syntax: `;`
- Does nothing(!)
- Always complete normally

Expression Definition

- An expression is a series of variables, operators, and method calls that evaluates to a single value

Expression Examples

- `counter = 0`
- `counter++`
- `reset(counter)`
- `counter + minValue`

Expression Statements

- `<expression> ;` is a `<statement>`
- Only some expressions
  - Assignment expressions
  - Any use of `++` or `--`
  - Method calls
  - Object creation expressions

Expression Statement Examples

- `counter = 0;`
- `counter++;`
- `reset(counter);`
- `counter + minValue;`

Block Definition

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

Empty Block

- Syntax: `{ }`
- Does nothing(!)
- Always complete normally

Block Example

```
{
   counter = 0;
   counter++;
   {
      reset(counter);
      counter+minValue;
   }
}
```

Conditional Statements

Conditional Statement Definition

- A conditional statement conditionally performs statements based on a criterion

Conditional Statements in Java

- Two statements
  - `if-else`
  - `switch-case`
- `if-else`
  - an alternative
- `switch-case`
  - a n-choice

If-else Statement

- Two forms
  - `if (expression) statement`
  - `if (expression) statement1 else statement2`
- Criterion
  - boolean expression
- Statements
  - simple
  - compound

If-else Example

```
if ( (number % 2) == 0) {
   System.out.println(number+" is an even
     number");
} else {
   System.out.println(number+" is an odd number");
}
```

Switch-case Statement

- One form

```
switch (<expression>) {
   case value: statement;
   potentially default: statement;
 }
```

- Criterion
  - char, byte, short, or int expression
- Statements
  - simple
  - compound

Switch-case Example

```
switch (number % 3)  {
   case 0:
     System.out.println(number+" can be divided by 3");
     break;
    case 1:
     System.out.println("Number%3=1");
     break;
   case 2:
     System.out.println("Number%3=2");
     break;
}
```

Let's Have a Break

```
switch (day)  {
   case 1:
   case 2:
   case 3:
   case 4:
   case 5:
     System.out.println("Working day");
     break;
   case 6:
   case 7:
     System.out.println("Having a rest!");
     break;
}
```

Default

```
switch (day)  {
   case 1:
   case 2:
   case 3:
   case 4:
   case 5:
     System.out.println(“Working day”);
     break;
   case 6:
   case 7:
     System.out.println(“Having a rest!”);
     break;
   default:
     System.out.println(“From which planet are you?”);
}
```

Loop Statements

Loop Statement Definition

- A loop statement iteratively performs statements

Loop Statements in Java

- Three statements
  - `while`
  - `do-while`
  - `for`

While Statement

- One form
  - `while (expression) statement`
- Criterion
  - boolean expression
- Statements
  - simple
  - compound

While Example

```
int counter = 0;
while (counter != 3) {
   System.out.println("Run "+counter);
   counter++;
}
```

Do-while Statement

- One form
  - `do statement while (expression);`
- Criterion
  - boolean expression
- Statements
  - simple
  - compound

Do-while Example

```
int counter = 0;
do {
   System.out.println("Run "+counter);
   counter++;
} while (counter !=3);
```

For Statement

- One form
  - `for (init; condition; increment) statement`
- Optional expressions
  - initialization
  - condition
  - increment
- Statements
  - simple
  - compound

For Example

```
int myValue = 10;
for (int i = 0; i < myValue; i+=2){
    System.out.println(myValue+" < "+i);
}
```

Branching Statements

Branching Statement Definition

- A branching statement alters the execution sequence

Branching Statements in Java

- Three statements
  - `return`
  - `break`
  - `continue`

Return Statement

- Two forms
  - `return;`
  - `return <value>;`
- Exits from the current method

Return Example

```
public int dividedByThree(int number) {
  switch (number % 3)  {
     case 0:
       System.out.println(number+" can be divided by 3");
       return number;
     case 1:
       System.out.println(number+"%3=1");
       break;
     case 2:
       System.out.println(number+"%3=2");
       break;
   }
   return 0;
}
```

Break Statement

- Two forms
  - `break;`
  - `break <label>;`
- Terminates statements
  - `switch`
  - `while`
  - `do-while`
  - `for`

Break Example

```
switch (number % 3)   {
    case 0:
       System.out.println(number+" can be divided by 3");
       break;
     case 1:
       System.out.println(number+"%3=1");
       break;
    case 2:
       System.out.println(number+"%3=2");
       break;
}
```

Break Example with Label

```
int first, second;
compute:
   for (int i = 0; i<10 ; i++){
      for (int j = 0; j< i; i++) {
         int sum = i+j;
         if ( sum > 15 ){
            first = i;
            second = j;
            break compute;
         }
      }
   }
System.out.println(“Found 15=”+first+”+”+second);
```

Continue Statement

- Two forms
  - `continue;`
  - `continue <label>;`
- Go back to iteration expressions of statements
  - `while`
  - `do-while`
  - `for`

Continue Example

```
for (int i = 0; i<100 ; i++){
      if ( (i % 7) != 0 )
         continue;
      System.out.println(i+" may be divided by 7");
}
```

Continue Example with Label

```
compute:
    for (int i = 0; i<10 ; i++){
        for (int j = 0; j< i; i++) {
            int sum = i+j;
            if ( sum > 15 ){
                System.out.println("15="+i+"+"+j);
                continue compute;
            }
        }
    }
```

Conclusions

Conclusions

- Variables
- Simple statements
  - expressions
  - blocks
- Control flow statements
  - conditional statements
  - loop statements
  - branching statements

Example

```
package pl.poznan.ae.compProg;

import java.util.*;

public class Sorter {
  private List _words;

  public void sort(String[] words){
    _words = Arrays.asList(words);
    Collections.sort(_words);
  }
  public String getSortedWords(){
    String sortedString = "";
    for (int i = 0; i< _words.size(); i++){
      sortedString += _words.get(i);
    }
    return sortedString;
  }
 public static void main(String[] args){
    Sorter sorter = new Sorter();
    sorter.sort(args);
    System.out.println(sorter.getSortedWords());
 }
}
```

See you next week