Ch 2. Java Programming Concepts and Structures

2.1 Java Reserved Keywords

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

* Reserved but not used (const, goto)

Keywords for Data Type Declarations

- Java has two fundamental data types: primitive and class object types.
- The primitive data types are divided into the following categories:
- Numeric data types keyword: **byte** (8-bit), **short** (16-bit), **int** (32-bit), **long** (64-bit), **float** (32-bit), **double** (64-bit)
- Character data type keyword: **char** (16-bit Unicode: '\u0000' to '\uFFFF'),
- Boolean data type keyword: **boolean** (8-bit)**, false, true**

Keywords for Flow Controls

- If, else
- for, while, do
- switch, case, break, default
- continue
- return
- try, catch, finally, throw

Class, Method, and Data Fields Qualifiers

- package
- import
- interface, class, abstract
- extends, implements
- private, protected, public, static, final, volatile
- synchronized, throws, transients, void, native, super

Keywords for Class Construction

package

- Used to create a class library of many classes

import

- It is used to import class libraries into a user program.

class

- The keyword **class** begins a class declaration and is followed by a class name
- It can be seen as a template for creating (instantiating) objects for use in a program

static

- Static methods and data fields (variables) are shared by all instances of a class
- static class methods are for operating on the class itself, rather than for specific class instances.
- static class fields (variables) are defined to be shared by all instances of a class.
- The main() of every Java stand-alone program is a static method.

extends

- For creating a subclass from super class (inheritance) or to extend the definition of a super or parent class.
- The subclass is called the derived class

interface

- Defines methods, like a class, but does not provide any implementations for those methods

implements

- Used to create a new class from an interface (a class with all abstract methods and/or abstract data)

abstract

- For identifying abstract classes and methods
- The implementation of an abstract class is deferred to its subclasses and may not be instantiated

final

- The final modifier indicates that a declared class may not be extended.

Variable and Constant Declarations

final

- The final modifier indicates that the declared variable may not be changed, therefore, it is treated as a read-only variable or a constant.

synchronized

- For marking synchronization method that control resource sharing among multiple threads

transient

- This modifier indicates that a variable may not be serialized
- A transient variable may not be declared as final or static

volatile

- For declaring a variable that may be accessed by several threads without synchronization

Access Control Modifiers

public

- Used to specify a *class* or *interface* to be accessed outside its package
- Used to enable a data field, method, or constructor to be accessed anywhere its class may be accessed.
- public class: a class may be accessed outside of its package.
- public interface: an interface may be accessed outside of its package.
- public field variable: a variable may be accessed anywhere that its class may be accessed.
- public method: a method may be accessed anywhere that its class may be accessed.
- public constructor
- public inner class: an inner class may be accesses anywhere its class may be accesses

protected

- Used to enable a data field, method, or constructor to be accessed by classes or interfaces of the same package; or by subclass of the class in which it is declared
- protected field variable
- protected methods
- protected constructors
- protected inner class

private

- Used to restrict a data field, method, or constructor to be accessed only within the class interfaces of the same package; or by subclass of the class in which it is declared
- private field variable

- private methods
- private constructors
- private inner class

Other Keywords

- Methods Declaration
  return // Return the program control
  void // Not return value
- Creating Object
  new // An operator for constructing an object or instance of its class
- Reference to a current running object
  this
- Determine Object Reference
  instanceof // Determine whether an object reference is an instance of the class, interface, or array type
- Inheritance
  super // Referring to its methods or variables of its parent class
  extends // for making a subclass
- Exception Handling
  try
  catch
  throw
  throws
  finally
- Mixed Language Programming
  native
  - The *native* keyword is used for identifying native methods written in C/C++.
  - The Java Native Interface (JNI) is then used to create a shared dynamic link library for use in mixed-language programming applications.

2.2 Java Data Types

Numeric data types:

byte - 1-byte (8-bit), signed small integer (-128 to 127)

short - 2-byte or16-bit signed integer (-32768 to 32767)

int - 4-byte or 32-bit signed integer (-2147483648 to 2147483647)

long - 8-byte or 64-bit signed integer

float - 4-byte (±38E, 32-bit), IEEE-754 standard

double - 8-byte (±308E, 64-bit), IEEE-754 standard

Character data type:

char (2 bytes Unicode; www.unicode.org)

Boolean data type:

boolean (1-byte: true, false)

2.3 Java Identifiers

A Java identifier is a name that uniquely identifies a variable, method, or class. The identifier naming restrictions are

- All identifiers must begin with a letter, an underscore (_), or a dollar sign ($)
- An identifier can include, but not begin with numbers
- An identifier cannot include a white space (tab, space, linefeed, or carriage return)
- Identifiers are case-sensitive
- Java keywords cannot be used as identifiers

There are two types of variables in Java:

- Primitive variables
- Object (Instance) variables

All primitive variables are defined within a class. There is no global variable in the Java. Every variable has

- a name: a valid identifier
- a type: boolean, byte, char, short, int, long, float, double
- a size
- a value

Variables

- An identifier is a variable name with a type and a valid identifier that consists of a sequence of letters and digits.
- Java sets no limits on the number of characters in a name.
- The first character must be a letter.
- The under score _ is considered a letter.
- A Java keyword cannot be used as a name.
- Uppercase and lowercase letters are distinct (case sensitive).
- Java allows explicit data type conversion between numeric types:
  byte -> short -> int -> long -> float -> double
- However, a boolean type cannot be converted to other data type.

```
double x = 10.90;
int n = (int) x;        // type demotion
```

Constant Variables

Constants are also called named constants or read-only variables.

A constant variable is created by placing the keyword **final** at the beginning of the declared variables, for example:

```
class thisClass
{ public static final double n = 12.00; }
```

Visibility and Lifetime of Variables

The scope rules for identifiers are as listed below. The rules determine when and where the variables are available for accessing:

- Class scope
  - Methods
  - Instance variables
- Block scope
  - Available only within a pair of the beginning and ending braces "{ }"
- Method scope
  - Local variables: variables declared in a method are available for access only within the method
  - Methods

Recommended Java Variable Naming Conventions:

Variable Naming

- Recommend that the first letter of a word is in lowercase:
  - wages
  - hourlyRate

Read-Only Variables

- All uppercase letter with an underscore to connect words.
  MAX_AGE
  MAX_COUNT
- Combining final and static

```
static final private int MIN = 0;
static final private int MAX = 10000;
```

Method Naming

- Method names are like variable names with the following naming conventions, namely, the first letter of each, except the first, word is capitalized:
  getName()
  readAddress()

Class Naming

- Standard class packages are given in lowercase
- The first letter of each word in the class name is capitalized
  - public String
  - public StringBuffer

Examples:

int counter;

Define firstName and lastName as references for referring to String type objects:

String firtstName;

String lastName;

Literal Constants

- **Integer literals**:

  Octal: `0, 01, 02, 03, 04, 05, 06, 07, 010, 011, 012, 013, 014, 015, 016, 017`

  Decimal: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15`

  Hexadecimal: `0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf or (0xA, 0xB, 0xC, 0xD, 0xE, 0xF)`

- **ASCII character literals**:

  `'a', 'b', … 'z'`
  `'A', 'B', .. 'Z'`
  `'0', '1', '2', …'9'`

- **String literals**:

  `"The Java programming Language"`
  `"W"`

- **Floating-point number literals**

  `123.0  0.0  0.1E-6`

- **Character escape sequence**

```
\b    backspace          \u0008
\t    tab                \u0009
\n    linefeed           \u000a
\r    carriage return    \u000d
\"    double quote       \u0022
\'    single quote       \u0027
\\    backslash          \u005c
\DDD  Octal character
\uxxx Unicode Character
```

- **System defined literals**

Decimal	Hex	Octal
0	0x0	0
1	0x1	01
2	0x2	02
3	0x3	03
4	0x4	04
5	0x5	05
6	0x6	06
7	0x7	07
8	0x8	010
9	0x9	011
10	0xA	012
11	0xB	013
12	0xC	014
13	0xD	015
14	0xE	016
15	0xF	017

Example 2-1: A Java example shows how to declare and initialize variables.

```
/* DataTypeEx1.java
 *
 * E:\LinJava\LinJavaExs\2_DataTypes>javac datatypeex1.java
 * E:\LinJava\LinJavaExs\2_DataTypes>java DataTypeEx1
 * Data Type Example
 * -----------------
 * letter_A = A
 * Max Count = 100
 * Tolerance = 1.0E-9
 */

public class DataTypeEx1
{  public static void main(String[] args)
   {
            String heading = "Data Type Example";
            String line = "-----------------";
            char letter_A = 'A';
            int max_count = 100;
            double tolerance = 0.000000001;
            System.out.println(heading);
            System.out.println(line);
            System.out.println("letter_A = " + letter_A);
            System.out.println("Max Count = " + max_count);
            System.out.println("Tolerance = " + tolerance);
   }
}
```

Example 2-2: A program displays the un-initialized variables.

```
/* UnInitVars.java
 *
 * This program displays default values of uninitiazlized variables.
 *
 * E:\LinJava\LinJavaExs\2_DataTypes>javac UnInitVars.java
 * E:\LinJava\LinJavaExs\2_DataTypes>java UnInitVars
 * Printing Uinitialized variables and Objects
 * ----------------------------------------
 * flag       = false
 * byte_val   = 0
 * char_val   =
 * short_val  = 0
 * int_val    = 0
 * long_val   = 0
 * float_val  = 0.0
 * double_val = 0.0
 * floatArray[3] = 0.0 0.0 0.0
 */
public class UnInitVars
{
      boolean     flag;
      byte        byte_val;
      char        char_val;
      short       short_val;
      int         int_val;
      long        long_val;
      float       float_val;
      double      double_val;

  public static void main(String[] args)
   {
      UnInitVars newApp = new UnInitVars();
      newApp.run();
    }
   void run()
   {
   float[] floatArray = new float[3];
   String heading = "Printing Uinitialized variables and Objects";
   String line    = "----------------------------------------";
   System.out.println(heading);
   System.out.println(line);
   System.out.println("flag       = " + flag);
   System.out.println("byte_val   = " + byte_val);
   System.out.println("char_val   = " + char_val);
   System.out.println("short_val  = " + short_val);
   System.out.println("int_val    = " + int_val);
   System.out.println("long_val   = " + long_val);
   System.out.println("float_val  = " + float_val);
   System.out.println("double_val = " + double_val);
   System.out.println("floatArray[3] = " + floatArray[0] + " "
                  + floatArray[1] + " " + floatArray[2]);
   }
}
```

2.4 Java Operators

Operators are normally used for the following purposed:

- Combining primitive expressions into complex expression
- Perform arithmetic operations
- Logic operation and reasoning
- Comparison and reasoning

Comments // --- one line comment

/* Beginning comment -- C style

*/ End of comment

/**

*/ -- for automatic documentation generation

Binary Arithmetic Operators

```
+     Addition operator:      int n = 10 + 20;
-     Subtraction operator:   int n = 20 - 10;
*     Multiplication:         int n = 20 * 10;
/     Division:               int n = 20 / 10;
%     Remainder or modulus:   int n = 1 % 2;
```

Unary Arithmetic Operators

```
+     Positive operand: +10
-     Negative operand: -10
++    Increment by 1:
            int num = 10;
            int n = ++num;    //pre-increment (n holds 11)
            int n = num++;    //post-increment (n holds 10)
--    Decrement by 1:
            int num = 10;
            int n = --num;    //pre-decrement (n holds 9)
            int n = num--;    //post-decrement (n holds 10)
```

The Bit-Wise Operators

```
~     Bit-wise NOT:     int num = ~0x01;
                        //(result 0xFE or 254)
&     Bit-wise AND:     int num = 0x02 & 0x01;
                        //(result 0x00)
|     Bit-wise OR:      int num = 0x02 | 0x01;
                        //(result 0x03)
^     Bit-wise EXOR:    int num = 0x0F ^ 0xF0;
                        //(result 0xFF or 255)
>>    Shift Right:      // Shift right one bit is the same
                        // as div by 2
      int num = 8 >> 2; //(num holds 2)
<<    Shift Left:       // Shift left one bit is the same
                        // as times 2
      int num = 8 << 2; //(num holds 32)
```

Assignment Operators

```
=     Assignment
+=    Add and assign
-=    Sub and assign
*=    Multiply and assign
/=    Divide and assign
%=    Remainder and assign
**=   Exponentiation and assign
<<=   Shift left and assign
>>=   Shift right and assign
&=    Bit-wise AND and assign
|=    Bit-wise OR and assign
^=    Bit-wise XOR and assign
```

Relational Operators

```
==    Equal
!=    Not equal
<     Less than
<=    Less than or equal
>     Greater than
>=    Greater than or equal
```

Logical Operators

```
&&    AND
||    OR
!     NOT
```

String concatenation operator

```
+
```

2.5 Flow Controls and Exception Handling

Control Statements

```
if(condition){
        statements;
        }

if(condition){
        statements;
        }
else{
        statements;
}

for(statement1; condition; statement2)
        { block statement;}
```

```
do{
        statements
        } while(condition);

switch(expression){
        case 1:
                statements;
                break;
        case 2:
                statements;
                break;
        …….
        case n:
                statements;
                break;
        default:
                statements;
                break;
}
```

Example 2-3: A for loop example prints upper case ASCII characters.

```
//forLoop.java
//
//E:\LinJavaExs\2_JavaConcepts>java forLoop
//
//Printing Upper Case ASCII Characters:
//
//AA     BB     CC     DD     EE     FF     GG     HH     II     JJ
//KK     LL     MM     NN     OO     PP     QQ     RR     SS     TT
//UU     VV     WW     XX     YY     ZZ
//
import java.io.*;
public class forLoop
{
  public static void main(String[] args)
 {
      System.out.println("\nPrinting Upper Case ASCII Characters:\n");
      for (char uChar = 'A'; uChar <= 'Z'; uChar++)
      {
      System.out.write(uChar); // print ASCII characters
      System.out.print(uChar); // print ASCII numbers
      System.out.print('\t');
      }
 }
}
```

2.6 Classes and Objects

Definitions:

- A class is a data type of an object and the object is an instance of its class.
- An object is a collection of fields or data values (properties or attributes) plus methods for manipulating data.
- Object creation: an object is created using the *new* operator, which invokes a constructor method of a class to initialize an new object as requested.

Class

- public methods (interfaces) as seen by clients
- Internal implementation
- Super class

java.lang.Object - the common superclass for all Java classes

Class Inheritance

- Reusing codes
- Classes inherits the instance variables and methods of the classes above them in the hierarchy
- A class can extend its inherited characteristics by adding instance variables and methods
- A class can extend its inherited characteristics by overriding inherited methods

Abstract Class

- Classes that must never be instantiated in a class hierarchy
- For defining features and behavior common to their subclasses
- In Java, the class Object is at the base or root of Java's class hierarchy

Protected Method

When a method should be visible to subclass but not to the rest of the system

Abstract Method

A method abstract (in an abstract class) when that method must be implemented by all subclasses

Final Method

Declare a method final when that method should be inherited but not overridden by any subclass

Super Method

When overriding a super-class method, use that super-class method (super)

Methods

main()

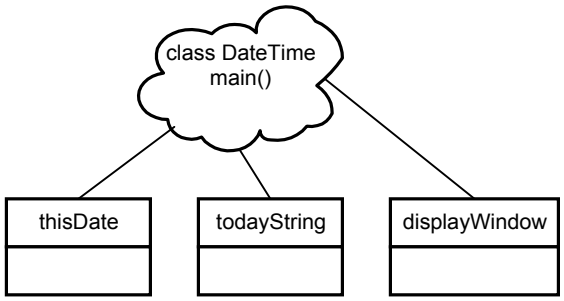
JVM sends the message main() to a program object

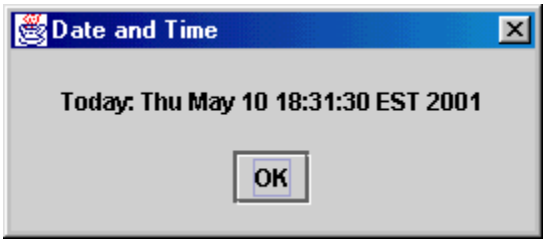
Execution of the method main()

Example 2-4: This example program does the following tasks to display system's date and time in a dialog window:

1. Create a user define class DateTime.
2. Create a Date instance called thisDate to obtain system's time and date information.
3. Create a String instance called today that contains time and date information in String format.
4. Use the showMessageDialog() method to display time and date in the dialog window.

A object diagram of this system is as shown below.





```
// DatTime.java
//
// This program is a date and time display applications.
// 1. Create a user define class DateTime.
// 2. Create a Date instance called thisDate to obtain system's time and
//    date information.
// 3. Create a String instance called todayString that contains time and
date
//    information in String format.
// 4. Use the showMessageDialog() method to display time and date
//    in the dialog window.
//
//
import javax.swing.JOptionPane;
import java.util.*;             // Date class

public class DateTime
{
  public static void main(String [] args)
  {
    Date thisDate = new Date();
    String todayString = thisDate.toString();

    JOptionPane.showMessageDialog(null, "Today: " + todayString, "Date
      and Time", JOptionPane.PLAIN_MESSAGE);

    System.exit(0);
  }
}
```

Example 2-5: A Java program uses class String.

```
/* StringTypeEx1.java
 *
 *
 */
public class StringTypeEx1
{  public static void main(String[] args)
   {
      String      msg = "Hello";
      String e = " ";         // Empty string
      e = "Set this string later";
      String m1 = "Computer";
      String m2 = "Electrical and Computer Engineering";
            System.out.println("msg = " + msg);
            System.out.println("e = " + e);
            System.out.println("m1 = " + m1);
            System.out.println("m2 = " + m2);

   }
}
```

Output

```
E:\LinJava\LinJavaExs\2_DataTypes>javac StringTypeEx1.java
E:\LinJava\LinJavaExs\2_DataTypes>java StringTypeEx1
msg = Hello
e = Set this string later
m1 = Computer
m2 = Electrical and Computer Engineering
```

2.7 The Math class

Methods of Math class are defined as the following format:

```
public static int abs(int aNumber)
public static long abs(long aNumber)
public static float abs(float aNumber)
public static double abs(float aNumber)
public static double exp(double aNumber)
public static double log(double aNumber)
public static int max(int aNumber, int bNumber)
public static long max(long aNumber, long bNumber)
public static float max(float aNumber, float bNumber)
public static double max(float aNumber, float bNumber)
public static int min(int aNumber, int bNumber)
public static long min(long aNumber, long bNumber)
public static float min(float aNumber, float bNumber)
public static double min(float aNumber, float bNumber)
public static double pow(double aNumber, double bNumber)
public static double random()
public static void srandom(double a)
public static int round(float aNumber)
public static int round(double aNumber)
public static double sin(double aNumber)
public static double cos(double aNumber)
public static double tan(double aNumber)
public static double asin(double aNumber)
public static double acos(double aNumber)
public static double atan(double aNumber)
public static double atan2(double aNumber)
```