Module 3

Identifiers, Keywords, and Types

Objectives

- Use comments in a source program
- Distinguish between valid and invalid identifiers
- Recognize Java technology keywords
- List the eight primitive types
- Define literal values for numeric and textual types
- Define the terms *primitive variable* and *reference variable*

Objectives

- Declare variables of class type
- Construct an object using `new`
- Describe default initialization
- Describe the significance of a reference variable
- State the consequences of assigning variables of class type

Relevance

- Do you know the primitive Java types?
- Can you describe the difference between variables holding primitive values as compared with object references?

Comments

The three permissible styles of comment in a Java technology program are:

```
// comment on one line

/* comment on one
 * or more lines
 */

/** documentation comment
 *  can also span one or more lines
 */
```

Semicolons, Blocks, and White Space

- A *statement* is one or more lines of code terminated by a semicolon (`;`):

```
totals = a + b + c
         + d + e + f;
```

- A *block* is a collection of statements bound by opening and closing braces:

```
{
  x = y + 1;
  y = x + 1;
}
```

Semicolons, Blocks, and White Space

- A *class* definition uses a special block:

```
public class MyDate {
  private int day;
  private int month;
  private int year;
}
```

- You can nest block statements.

```
while ( i < large ) {
  a = a + i;
  // nested block
  if ( a == max ) {
    b = b + a;
    a = 0;
  }
  i = i + 1;
}
```

Semicolons, Blocks, and White Space

- Any amount of *white space* is permitted in a Java program.

  For example:

  ```
  {int x;x=23*54;}
  ```

  is equivalent to:

  ```
  {
    int x;

    x = 23 * 54;
  }
  ```

Identifiers

Identifiers have the following characteristics:

- Are names given to a variable, class, or method
- Can start with a Unicode letter, underscore (_), or dollar sign ($)
- Are case-sensitive and have no maximum length
- Examples:

```
identifier
userName
user_name
_sys_var1
$change
```

Java Programming Language Keywords

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

Reserved literal words: `null`, `true`, and `false`

Primitive Types

The Java programming language defines eight primitive types:

- Logical – `boolean`
- Textual – `char`
- Integral – `byte`, `short`, `int`, and `long`
- Floating – `double` and `float`

Logical – `boolean`

The `boolean` primitive has the following characteristics:

- The `boolean` data type has two literals, `true` and `false`.
- For example, the statement:

  ```
  boolean truth = true;
  ```

  declares the variable `truth` as `boolean` type and assigns it a value of `true`.

Textual – `char`

The textual `char` primitive has the following characteristics:

- Represents a 16-bit Unicode character
- Must have its literal enclosed in single quotes (`'  '`)
- Uses the following notations:

`'a'`	The letter `a`
`'\t'`	The tab character
`'\u????'`	A specific Unicode character, `????`, is replaced with exactly four hexadecimal digits . For example, `'\u03A6'` is the Greek letter phi [Φ].

Textual – `String`

The textual `String` type has the following characteristics:

- Is not a primitive data type; it is a class
- Has its literal enclosed in double quotes (`"  "`)

```
"The quick brown fox jumps over the lazy dog."
```

- Can be used as follows:

```
String greeting = "Good Morning !! \n";
String errorMessage = "Record Not Found !";
```

Integral – `byte`, `short`, `int`, and `long`

The integral primitives have the following characteristics:

- Integral primates use three forms: Decimal, octal, or hexadecimal

`2`	The decimal form for the integer `2`.
`077`	The leading `0` indicates an octal value.
`0xBAAC`	The leading `0x` indicates a hexadecimal value.

- Literals have a default type of `int`.
- Literals with the suffix `L` or `l` are of type `long`.

Integral – `byte`, `short`, `int`, and `long`

- Integral data types have the following ranges:

Integer Length	**Name or Type**	**Range**
8 bits	`byte`	-2^{7} to $2^{7}-1$
16 bits	`short`	-2^{15} to $2^{15}-1$
32 bits	`int`	-2^{31} to $2^{31}-1$
64 bits	`long`	-2^{63} to $2^{63}-1$

Floating Point – `float` and `double`

The floating point primitives have the following characteristics:

- Floating-point literal includes either a decimal point or one of the following:
  - `E` or `e` (add exponential value)
  - `F` or `f` (`float`)
  - `D` or `d` (`double`)

`3.14`	A simple floating-point value (a `double`)
`6.02E23`	A large floating-point value
`2.718F`	A simple `float` size value
`123.4E+306D`	A large `double` value with redundant D

Floating Point – `float` and `double`

- Literals have a default type of `double`.
- Floating-point data types have the following sizes:

Float Length	**Name or Type**
32 bits	`float`
64 bits	`double`

Variables, Declarations, and Assignments

```java
public class Assign {
  public static void main (String args []) {
    // declare integer variables
    int x, y;
    // declare and assign floating point
    float z = 3.414f;
    // declare and assign double
    double w = 3.1415;
    // declare and assign boolean
    boolean truth = true;
    // declare character variable
    char c;
    // declare String variable
    String str;
    // declare and assign String variable
    String str1 = "bye";
    // assign value to char variable
    c = 'A';
    // assign value to String variable
    str = "Hi out there!";
    // assign values to int variables
    x = 6;
    y = 1000;
  }
}
```

Java Reference Types

- In Java technology, beyond primitive types all others are reference types.
- A *reference variable* contains a *handle* to an object.
- For example:

```
public class MyDate {
  private int day = 1;
  private int month = 1;
  private int year = 2000;
  public MyDate(int day, int month, int year) { ... }
  public String toString() { ... }
}
```

```
public class TestMyDate {
  public static void main(String[] args) {
    MyDate today = new MyDate(22, 7, 1964);
  }
}
```

Constructing and Initializing Objects

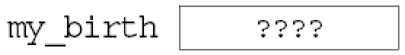
- Calling `new Xyz()` performs the following actions:
  a. Memory is allocated for the object.
  b. Explicit attribute initialization is performed.
  c. A constructor is executed.
  d. The object reference is returned by the `new` operator.
- The reference to the object is assigned to a variable.
- An example is:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

Memory Allocation and Layout

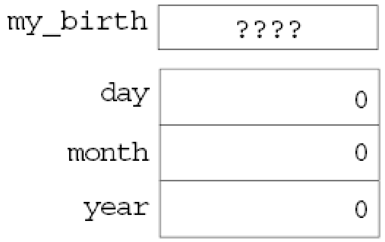
- A declaration allocates storage only for a reference:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```



- Use the `new` operator to allocate space for `MyDate`:

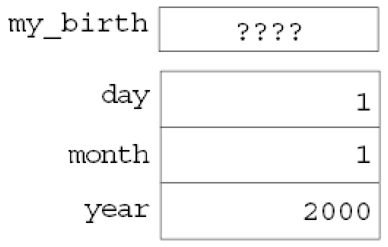
```
MyDate my_birth = new MyDate(22, 7, 1964);
```



Explicit Attribute Initialization

- Initialize the attributes as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

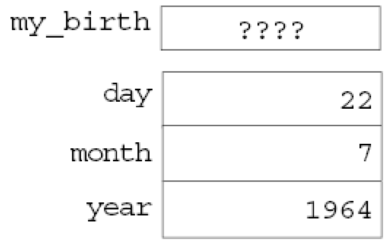


- The default values are taken from the attribute declaration in the class.

Executing the Constructor

- Execute the matching constructor as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

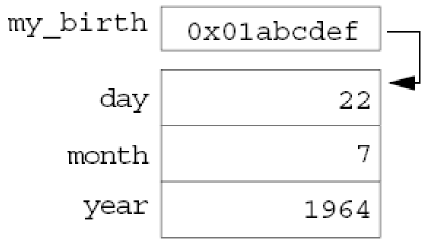


- In the case of an overloaded constructor, the first constructor can call another.

Assigning a Variable

- Assign the newly created object to the reference variable as follows:

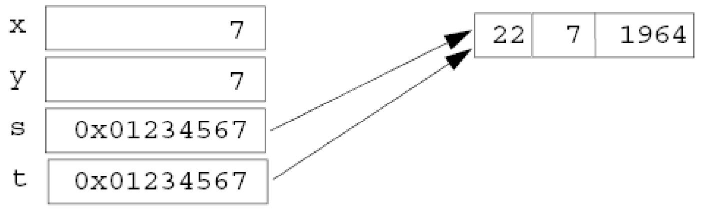
```
MyDate my_birth = new MyDate(22, 7, 1964);
```



Assigning References

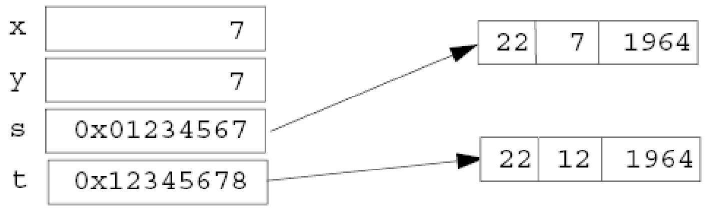
- Two variables refer to a single object:

```
int x = 7;
int y = x;
MyDate s = new MyDate(22, 7, 1964);
MyDate t = s;
```



- Reassignment makes two variables point to two objects:

```
t = new MyDate(22, 12, 1964);
```



Pass-by-Value

- In a single virtual machine, the Java programming language only passes arguments by value.
- When an object instance is passed as an argument to a method, the value of the argument is a *reference* to the object.
- The *contents* of the object can be changed in the called method, but the original object reference is never changed.

Pass-by-Value

```
public class PassTest {

  // Methods to change the current values
  public static void changeInt(int value) {
    value = 55;
  }
  public static void changeObjectRef(MyDate ref) {
    ref = new MyDate(1, 1, 2000);
  }
  public static void changeObjectAttr(MyDate ref){
    ref.setDay(4);
  }
```

Pass-by-Value

```

    public static void main(String args[]) {
      MyDate date;
      int val;

      // Assign the int
      val = 11;
      // Try to change it
      changeInt(val);
      // What is the current value?
      System.out.println("Int value is: " + val);
```

The result of this output is:

```
Int value is: 11
```

Pass-by-Value

```

    // Assign the date
    date = new MyDate(22, 7, 1964);
    // Try to change it
    changeObjectRef(date);
    // What is the current value?
    System.out.println("MyDate: " + date);
```

The result of this output is:

```
MyDate: 22-7-1964
```

Pass-by-Value

```

        // Now change the day attribute
        // through the object reference
        changeObjectAttr(date);
        // What is the current value?
        System.out.println("MyDate: " + date);
    }
}
```

The result of this output is:

```
MyDate: 4-7-1964
```

The `this` Reference

Here are a few uses of the `this` keyword:

- To resolve ambiguity between instance variables and parameters
- To pass the current object as a parameter to another method or constructor

The `this` Reference

```java
public class MyDate {
  private int day = 1;
  private int month = 1;
  private int year = 2000;

  public MyDate(int day, int month, int year) {
    this.day   = day;
    this.month = month;
    this.year  = year;
  }
  public MyDate(MyDate date) {
    this.day   = date.day;
    this.month = date.month;
    this.year  = date.year;
  }
```

The `this` Reference

```

  public MyDate addDays(int moreDays) {
    MyDate newDate = new MyDate(this);
    newDate.day = newDate.day + moreDays;
    // Not Yet Implemented: wrap around code...
    return newDate;
  }
  public String toString() {
    return "" + day + "-" + month + "-" + year;
  }
}
```

The `this` Reference

```
public class TestMyDate {
  public static void main(String[] args) {
    MyDate my_birth = new MyDate(22, 7, 1964);
    MyDate the_next_week = my_birth.addDays(7);

    System.out.println(the_next_week);
  }
}
```

Java Programming Language Coding Conventions

- Packages:

  `com.example.domain;`

- Classes, interfaces, and `enum` types:

  `SavingsAccount`

- Methods:

  `getAccount()`

- Variables:

  `currentCustomer`

- Constants:

  `HEAD_COUNT`

Java Programming Language Coding Conventions

- Control structures:

```
if ( condition ) {
  statement1;
} else {
  statement2;
}
```

- Spacing:
  - Use one statement per line.
  - Use two or four spaces for indentation.
- Comments:
  - Use // to comment inline code.
  - Use `/** documentation */` for class members.

สรุปท้ายบท

- เขียน method main ให้ถูกต้อง มิฉะนั้น ฟ้อง runtime แต่ args เปลี่ยนได้ และ public กับ static สลับหน้าหลังได้
- main เรียกใช้ instance variable หรือ method ได้ แต่ instance variable, method ต้องประกาศเป็น static เพราะการเรียกตรงจาก main ต้องเรียก ตัวที่ประกาศเป็น static เท่านั้น
- keyword เหมือน reserved word (reserved words **รวม** true, false, null)

สรุปท้ายบท

- **keyword** เป็น lowercase ทั้งหมด และที่เป็นแต่ไม่ใช้คือ goto และ const
- **primitive** มี 8 แต่มีแค่ 6 ตัวที่มีค่าเป็น positive และ negative
- ไม่สามารถใช้ **keyword** เป็นชื่อ **class** ได้ (คือ ถ้าใช้จะ compile ผ่านแต่จะ run ไม่ผ่าน โดยจะแสดง NoClassDefFoundError)

สรุปท้ายบท

- สำหรับ **byte** มีค่าตั้งแต่ -128 ถึง 127
- **char** เป็น unsigned 16 bits มีค่าอยู่ในช่วง 0 ถึง 65535

สรุปท้ายบท

- ค่าที่เป็นไปได้ของ char คือ 65, 041 เป็นต้น

```
Example :  class X{
              public static void main(String[] bc) {
                    char a = 65; char b = 0101;
                    char c = '\u0041';
                    char d = 0x0041; char e = 'A';
                    System.out.println(""+a+b+c+d+e);//AAAAA
                  }
            }
```

สรุปท้ายบท

- **แบบของตัวเลขโดยปริยาย**(Default number) ถ้าเป็นจำนวนเต็มคือ int และมีทศนิยมคือ double ควรระวังเรื่องขนาด
  Example :

```
wrong :  float f = 1.5;
right :     float f = 1.5f;
right :     float f = 1.5F;
```

สรุปท้ายบท

- เมื่อ b เป็น byte โดย b = 1 + 1 จะไม่มีปัญหา แต่

  b = 100 + 100; จะ compile error เรื่อง loss of precision

  Example :

```
wrong : byte b = 100 + 100;
wrong : byte b = (byte)100 + 100;
right : byte b = (byte)(100 + 100);
```

สรุปท้ายบท

Example :

```
wrong : byte b = 128;

right : byte b = (byte)128;
Compilation Ok but print -128

right : byte b = (byte)256;
Compilation Ok but print 0
```

สรุปท้ายบท

Example :

```
      right : char c = ‘A’;
              int  y = c;
     Compilation Ok and Print 65

     wrong : int  y = 65;
             char x = y;
     Compilation error:
     Found int,required char.
     right : char x = (char) y;
```

สรุปท้ายบท

- **String** รับ null ได้ แต่รับ Null ไม่ได้ และ null ไม่ใช่ keyword
- **String** รับ char, 'a', '\u1111' หรือ (String)'a' ก็ไม่ได้