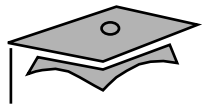


# Module 3

## 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;  
}
```



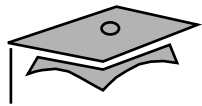
# Semicolons, Blocks, and White Space

- 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.

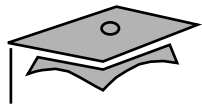
For example:

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

is equivalent to:

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



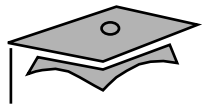


# Identifiers

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

---

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

---

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



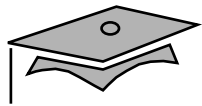
## Logical – boolean

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

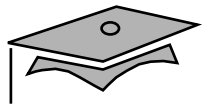
The textual `String` type has the following characteristics:

- 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`.

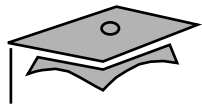


## Integral – byte, short, int, and long

- 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

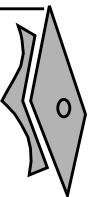
---



## 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

```
1  public class Assign {
2      public static void main (String args []) {
3          // declare integer variables
4          int x, y;
5          // declare and assign floating point
6          float z = 3.414f;
7          // declare and assign double
8          double w = 3.1415;
9          // declare and assign boolean
10         boolean truth = true;
11         // declare character variable
12         char c;
13         // declare String variable
14         String str;
15         // declare and assign String variable
16         String str1 = "bye";
17         // assign value to char variable
18         c = 'A';
19         // assign value to String variable
20         str = "Hi out there!";
21         // assign values to int variables
22         x = 6;
23         y = 1000;
24     }
25 }
```



## 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:

```
1  public class MyDate {  
2      private int day = 1;  
3      private int month = 1;  
4      private int year = 2000;  
5      public MyDate(int day, int month, int year) { ... }  
6      public String toString() { ... }  
7  }
```

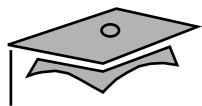
```
1  public class TestMyDate {  
2      public static void main(String[] args) {  
3          MyDate today = new MyDate(22, 7, 1964);  
4      }  
5  }
```



## 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);
```

my_birth	????
----------	------

- Use the new operator to allocate space for MyDate:

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

my_birth	????
day	0
month	0
year	0



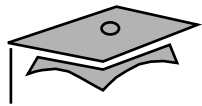
## Explicit Attribute Initialization

- Initialize the attributes as follows:

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

my_birth	????
day	1
month	1
year	2000

- 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);
```

my_birth	????
day	22
month	7
year	1964

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

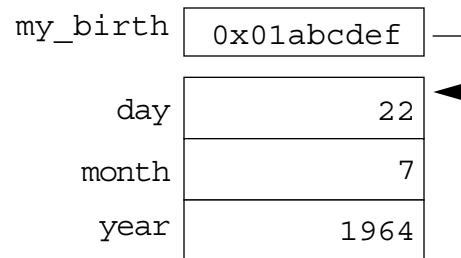


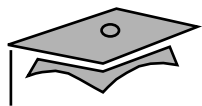


## Assigning a Variable

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

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

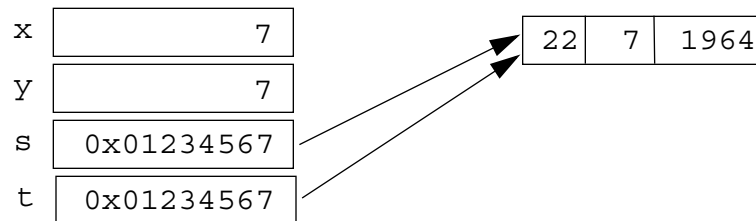




## Assigning References

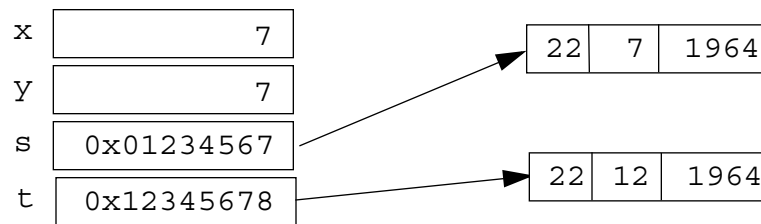
- Two variables refer to a single object:

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



- Reassignment makes two variables point to two objects:

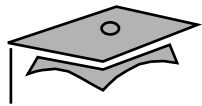
```
5  t = new MyDate(22, 12, 1964);
```





## 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

```
1  public class PassTest {  
2  
3      // Methods to change the current values  
4      public static void changeInt(int value) {  
5          value = 55;  
6      }  
7      public static void changeObjectRef(MyDate ref) {  
8          ref = new MyDate(1, 1, 2000);  
9      }  
10     public static void changeObjectAttr(MyDate ref) {  
11         ref.setDay(4);  
12     }
```



## Pass-by-Value

```
13
14     public static void main(String args[]) {
15         MyDate date;
16         int val;
17
18         // Assign the int
19         val = 11;
20         // Try to change it
21         changeInt(val);
22         // What is the current value?
23         System.out.println("Int value is: " + val);
```

The result of this output is:

Int value is: 11



## Pass-by-Value

```
24
25     // Assign the date
26     date = new MyDate(22, 7, 1964);
27     // Try to change it
28     changeObjectRef(date);
29     // What is the current value?
30     System.out.println("MyDate: " + date);
```

The result of this output is:

MyDate: 22-7-1964



## Pass-by-Value

```
31
32     // Now change the day attribute
33     // through the object reference
34     changeObjectAttr(date);
35     // What is the current value?
36     System.out.println("MyDate: " + date);
37 }
38 }
```

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





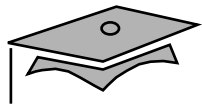
## The `this` Reference

```
1  public class MyDate {  
2      private int day = 1;  
3      private int month = 1;  
4      private int year = 2000;  
5  
6      public MyDate(int day, int month, int year) {  
7          this.day    = day;  
8          this.month  = month;  
9          this.year   = year;  
10     }  
11     public MyDate(MyDate date) {  
12         this.day    = date.day;  
13         this.month  = date.month;  
14         this.year   = date.year;  
15     }
```



## The `this` Reference

```
16
17     public MyDate addDays(int moreDays) {
18         MyDate newDate = new MyDate(this);
19         newDate.day = newDate.day + moreDays;
20         // Not Yet Implemented: wrap around code...
21         return newDate;
22     }
23     public String toString() {
24         return "" + day + "-" + month + "-" + year;
25     }
26 }
```



## The `this` Reference

```
1  public class TestMyDate {  
2      public static void main(String[] args) {  
3          MyDate my_birth = new MyDate(22, 7, 1964);  
4          MyDate the_next_week = my_birth.addDays(7);  
5  
6          System.out.println(the_next_week);  
7      }  
8  }
```



# 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.