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 *primitive variable* and *reference variable*
- Describe **constructing** and **initializing** objects
- State the result of assigning variables of class type

Statements

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;
}
```

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.

Identifiers and Keywords

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`

Restricted Keywords (used in Java Module as keywords , Since Java 9):

`open`, `module`, `requires`, `transitive`, `exports`, `opens`, `to`, `uses`, `provides` , `with`

Special usage words:

`var` (Since Java 11) , `yield` (Since Java 14)

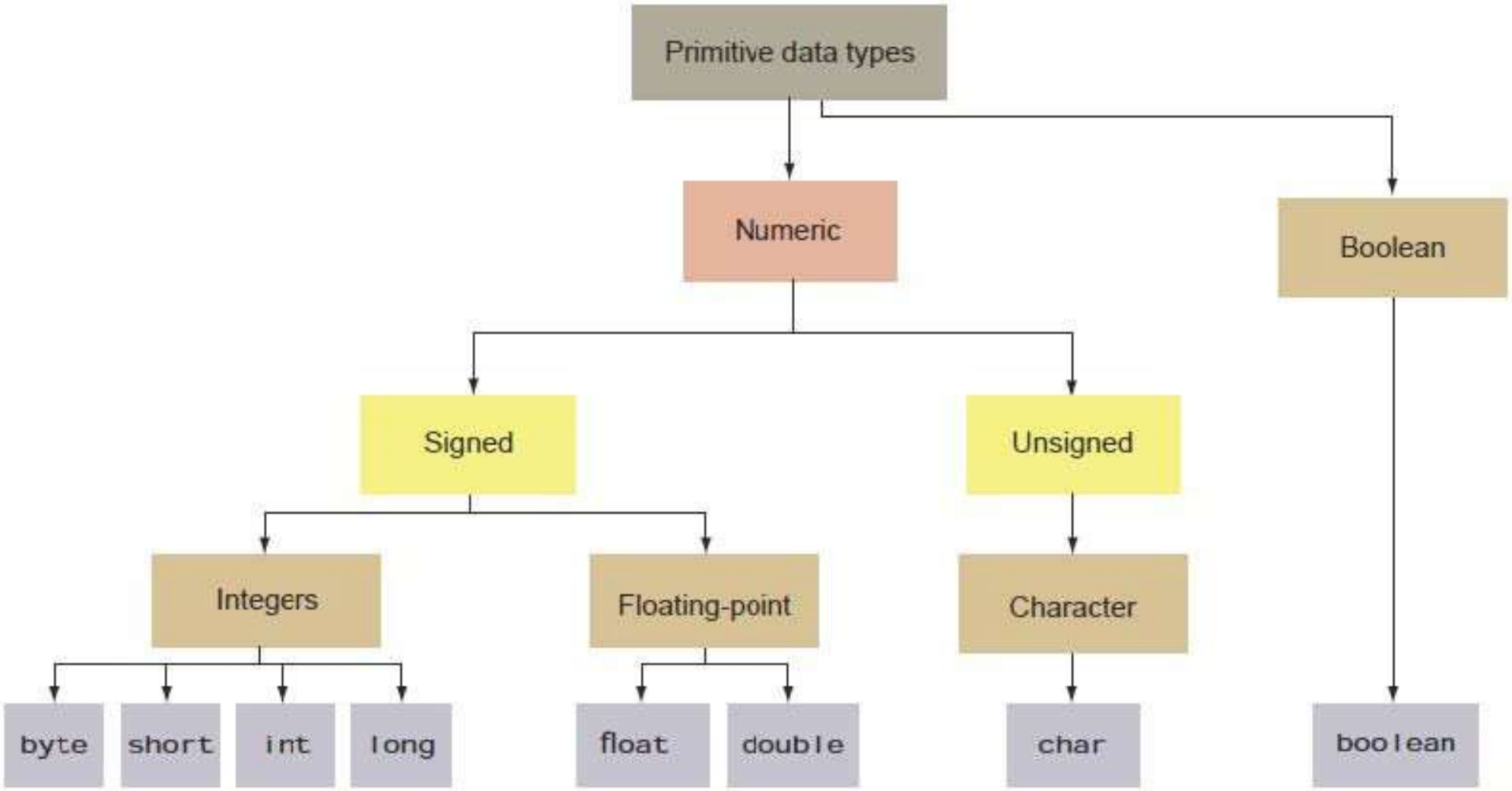
Data Types:

Primitive Types and Reference Types

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].

Java Escape Sequences

	Name	Unicode Code
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000A
\f	Formfeed	\u000C
\r	Carriage Return	\u000D
\\	Backslash	\u005C
\’	Single Quote	\u0027
\”	Double Quote	\u0022

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 4 forms: Decimal, binary, octal, hexadecimal

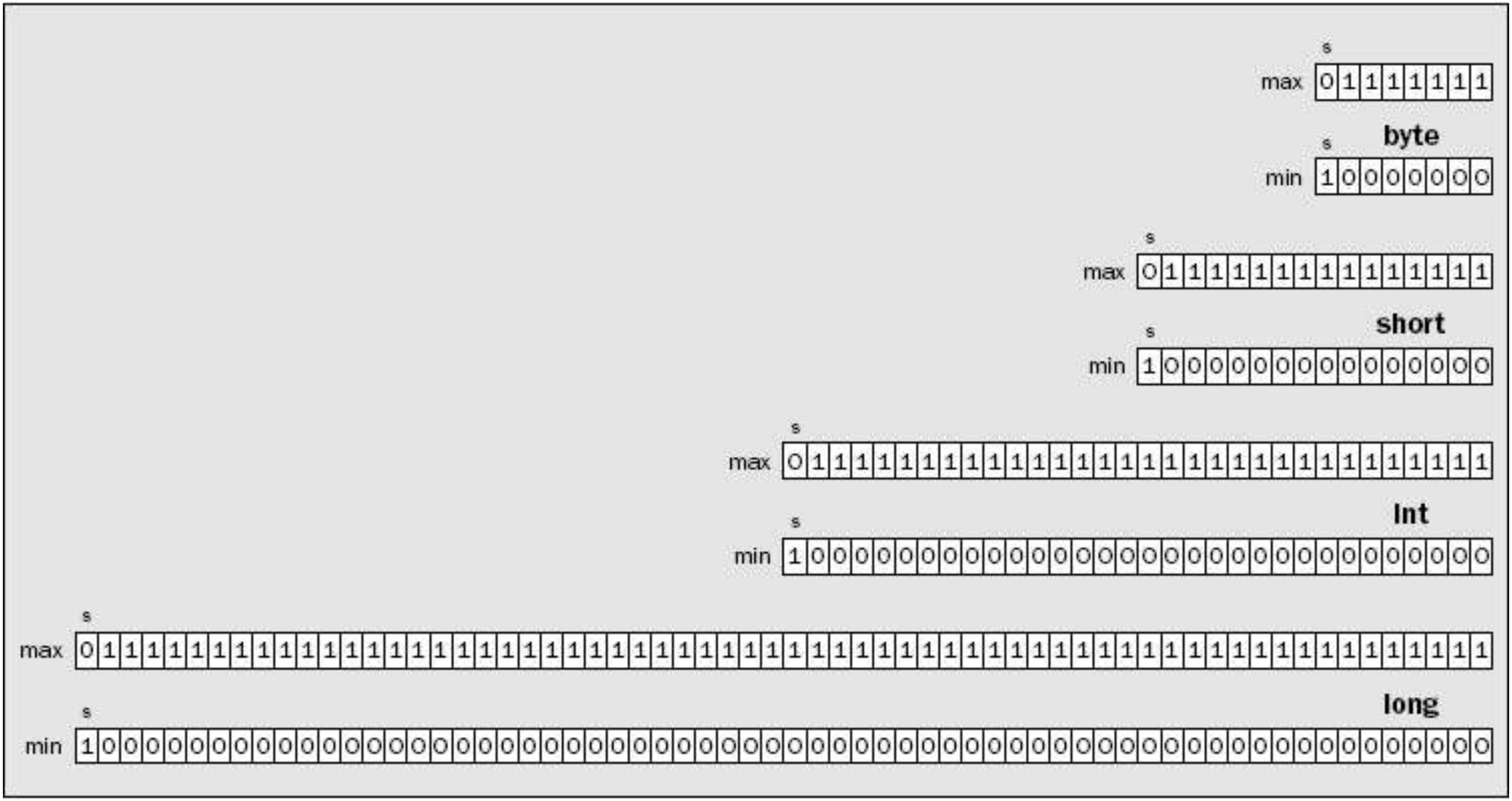
`26`	The decimal form for the integer `26`.
0b11010	The binary form for 26. (Since JDK 7).
`077`	The leading `0` indicates an octal value.
`0xBAAC`	The leading `0x` indicates a hexadecimal value.

- 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`

Using Underscore Characters in Numeric Literals

- Since Java SE 7 , underscore characters (_) can be used to separate groups of digits in a numerical literal. This can improve the readability of your code.

```
long creditCardNumber =1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =  3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

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,
4       1964);
5     }
```

Constructing and Initializing Objects

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);
```

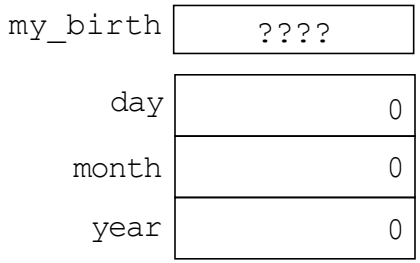
a. Memory Allocation and Layout

- A declaration allocates storage only for a reference in stack:

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

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

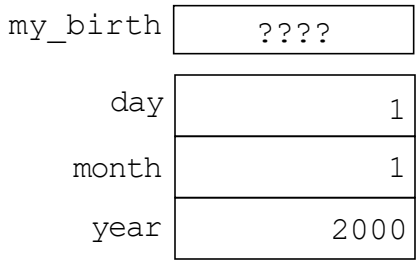
```
MyDate my_birth = new MyDate(22, 7, 1964);
```



b. Explicit Attribute Initialization

- Initialize the attributes as follows:

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

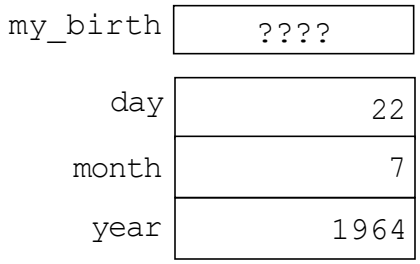


- The default values are taken from the attribute declaration in the class.

c. Executing the Constructor

- Execute the matching constructor as follows:

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

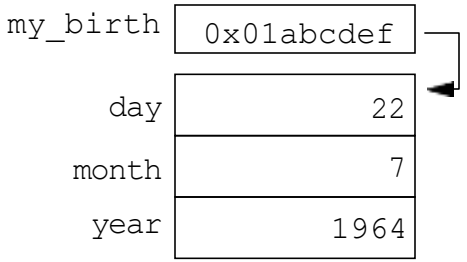


- In the case of an overloaded constructor, the first constructor can call another.

d. Assigning a Variable

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

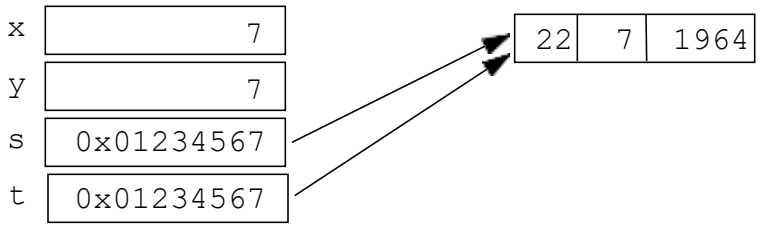
```
MyDate my_birth = new MyDate(22, 7, 1964);
```



Assigning References

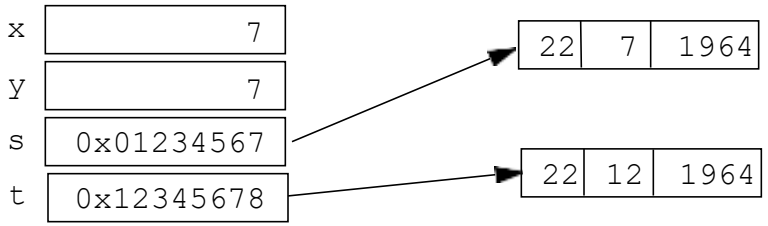
- Two variables refer to a single object:

```
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

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.

Summary

- Statements, Comments
- Coding Conventions
- Identifiers and Keywords
- Primitive Types and Reference Types
- Variables
- Constructing and Initializing Objects
- Pass-by-Value