4

Exploring Primitive Data types and Operators

Objectives

After completing this lesson, you should be able to do the following:

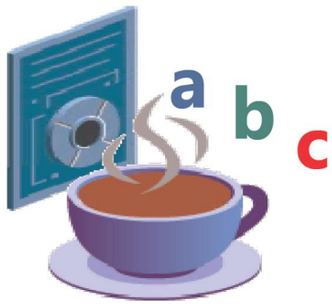
- Distinguish between reserved words and other names in Java
- Describe Java primitive data types and variables
- Declare and initialize primitive variables
- Use operators to manipulate primitive variables
- Describe uses of literals and Java operators
- Identify valid operator categories and operator precedence
- Use string object literals and the concatenation operator

Keywords and Reserved Words

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

Variable Types

- Eight primitive data types:
  - Six numeric types
  - One character type
  - One Boolean type (for truth values)
- User-defined types:
  - Classes
  - Interfaces
  - Arrays



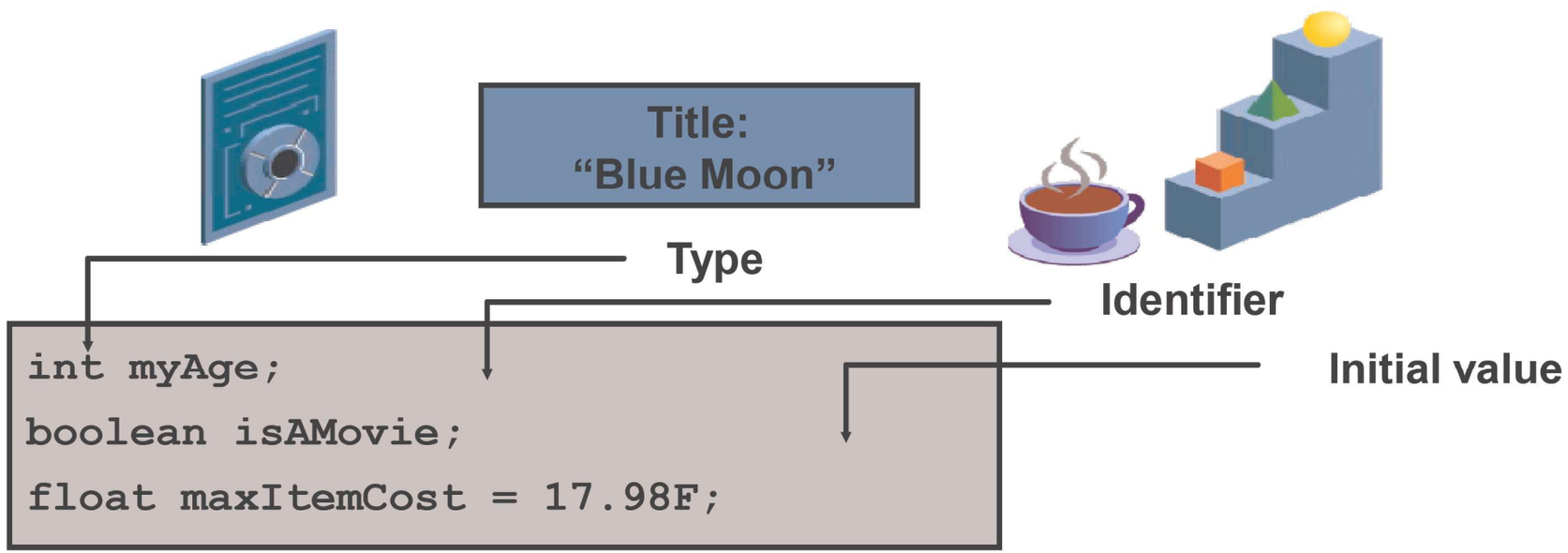
Primitive Data Types

Integer	Floating Point	Character	True False
byte short int long	float double	char	boolean
1, 2, 3, 42 07 0xff	3.0F .3337F 4.022E23	'a' '\141' '\u0061' '\n'	true false
0	0.0f	'\u0000'	false

Append uppercase or lowercase “L” or “F” to the number to specify a long or a float number.

Variables

- A variable is a basic unit of storage.
- Variables must be explicitly declared.
- Each variable has a type, an identifier, and a scope.
- There are three types of variables: class, instance, and method.



Declaring Variables

- Basic form of variable declaration:
  - `type identifier [ = value];`

```
public static void main(String[] args) {
    int itemsRented = 1;
    float itemCost;
    int i, j, k;
    double interestRate;
}
```

- Variables can be initialized when declared.

Local Variables

- Local variables are defined only within a method or code block.
- They must be initialized before their contents are read or referenced.

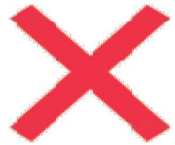
```
class Rental {
  private int instVar;       // instance variable
  public void addItem() {
    float itemCost = 3.50F; // local variable
    int numOfDays  = 3;     // local variable
  }
}
```

Defining Variable Names

- Variable names must start with a letter of the alphabet, an underscore, or a $ symbol.
- Other characters may include digits.

```
a              item_Cost
itemCost       _itemCost
item$Cost      itemCost2
```

```
item#Cost      item-Cost item*Cost
               abstract 2itemCost
```



- Use meaningful names for variables, such as `customerFirstName` and `ageNextBirthday`.

Numeric Literals

Six types: `byte`, `short`, `int`, `long`, `float`, `double`

Integer literals

```
0   1   42   -23795          (decimal)
02   077   0123              (octal)
0x0   0x2a   0X1FF            (hex)
365L   077L   0x1000L         (long)
```

Floating-point literals

```
1.0   4.2   .47
1.22e19   4.61E-9
6.2f   6.21F
```

Nonnumeric Literals

Boolean literals

```
true false
```

Character literals

```
'a'  '\n'  '\t'  '\077' '\u006F'
```

String literals

```
"Hello, world\n"
```

enum

- New feature introduced in Java SE 5.0
- Compiler support for Typesafe Enum pattern
- Looks like traditional `enum` (C, C++, Pascal)
  - `enum Season {WINTER, SPRING, SUMMER, FALL}`
- Very powerful

Operators

- Operators manipulate data and objects.
- Operators take one or more arguments and produce a value.
- There are 44 different operators.
- Some operators change the value of the operand.

Categories of Operators

There are five types of operators:

- Assignment
- Arithmetic
- Integer bitwise
- Relational
- Boolean

Using the Assignment Operator

The result of an assignment operation is a value and can be used whenever an expression is permitted.

- The value on the right is assigned to the identifier on the left:

```
int var1 = 0, var2 = 0;
var1 = 50;          // var1 now equals 50
var2 = var1 + 10; // var2 now equals 60
```

- The expression on the right is always evaluated before the assignment.
- Assignments can be strung together:

```
var1 = var2 = var3 = 50;
```

Working with Arithmetic Operators

- Used to perform basic arithmetic operations
- Work on numeric variables and literals

```
int a, b, c, d, e;
a = 2 + 2;    // addition
b = a * 3;    // multiplication
c = b - 2;    // subtraction
d = b / 2;    // division
e = b % 2;    // returns the remainder of division
```

More About Arithmetic Operators

Most operations result in `int` or `long`:

- `byte`, `char`, and `short` values are promoted to `int` before the operation.
- If either argument is of the `long` type, then the other is also promoted to `long`, and the result is of the `long` type.

```
byte b1 = 1, b2 = 2, b3;
b3 = b1 + b2;      // ERROR: result is an int
                   // b3 is byte
```

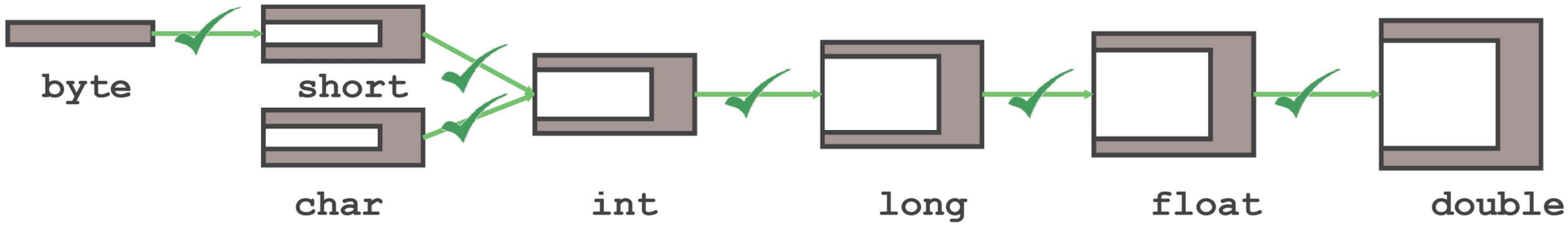
Guided Practice: Declaring Variables

Find the errors in this code and fix them:

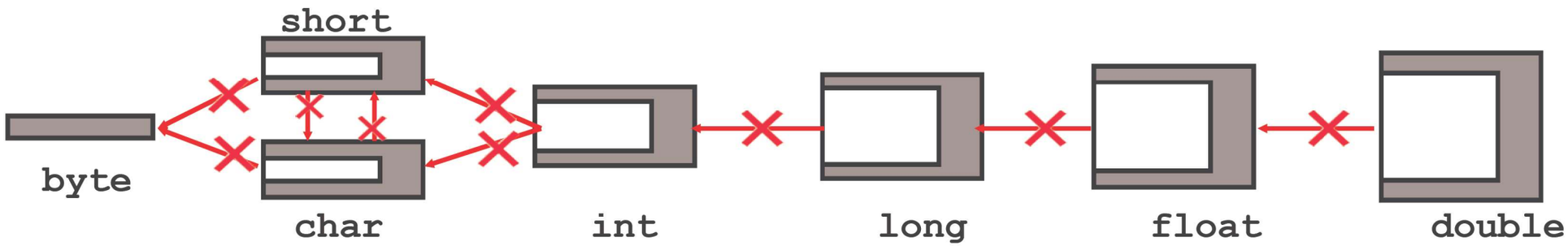
```
1   byte sizeof = 200;
2   short mom = 43;
3   short hello mom;
4   int big = sizeof * sizeof * sizeof;
5   long bigger = big + big + big     // ouch
6   double old = 78.0;
7   double new = 0.1;
8   boolean consequence = true;
9   boolean max = big > bigger;
10  char maine = "New England state";
11  char ming = 'd';
```

Examining Conversions and Casts

- Java automatically converts a value of one numeric type to a larger type (widening primitive conversions).



- Java does not automatically “downcast” (narrowing primitive conversion).



Incrementing and Decrementing Values

- The ++ and -- operators increment and decrement by 1, respectively:

```
int var1 = 3;
var1++;           // var1 now equals 4
```

- The ++ and -- operators can be used in two ways:

```
int var1 = 3, var2 = 0;
var2 = ++var1;    // Prefix:  Increment var1 first,
                  //           then assign to var2.
var2 = var1++;    // Postfix: Assign to var2 first,
                  //           then increment var1.
```

Relational and Equality Operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
int var1 = 7, var2 = 13;
boolean res = true;
res = (var1 == var2);      // res now equals false
res = (var2 > var1);       // res now equals true
```

Conditional Operator (? :)

- Useful alternative to `if…else`:

```
boolean_expr ? expr1 : expr2
```

- If *`boolean_expr`* is `true`, the result is *`expr1`*; otherwise, the result is *`expr2`*:

```
int val1 = 120, val2 = 0;
int highest;
highest = (val1 > val2) ? val1 : val2;
System.out.println("Highest value is " + highest);
```

Logical Operators

Results of Boolean expressions can be combined by using logical operators:

`&&` `&`	**AND (with or without short-circuit evaluation)**
`||` `|`	**OR (with or without short-circuit evaluation)**
`^`	**exclusive OR**
`!`	**NOT**

```
int var0 = 0, var1 = 1, var2 = 2;
boolean res = true;
highest = (val1 > val2)? val1 : val2;
res = !res;
```

Compound Assignment Operators

An assignment operator can be combined with any conventional binary operator:

```
double total=0, num = 1;
double percentage = .50;
…
total  = total + num;    // total is now  1
total += num;            // total is now  2
total -= num;            // total is now  1
total *= percentage;     // total is now .5
total /= 2;             // total is now 0.25
num %= percentage;      // num is now 0
```

Operator Precedence

Order	Operators	Comments	Assoc.
1	++ -- + - ~ ! (*type*)	Unary operators	R
2	* / %	Multiply, divide, remainder	L
3	+ - +	Add, subtract, add string	L
4	<< >> >>>	Shift (>>> is zero-fill shift)	L
5	< > <= >= instanceof	Relational, type compare	L
6	== !=	Equality	L
7	&	Bit/logical AND	L
8	^	Bit/logical exclusive OR	L
9	|	Bit/logical inclusive OR	L
10	&&	Logical AND	L
11	||	Logical OR	L
12	?:	Conditional operator	R
13	= *op*=	Assignment operators	R

More About Operator Precedence

- Operator precedence determines the order in which operators are executed:

```
int var1 = 0;
var1 = 2 + 3 * 4;     // var1 now equals 14
```

- Operators with the same precedence are executed from left to right (see

```
int var1 = 0;
var1 = 12 - 6 + 3;    // var1 now equals 9
```

Use parentheses to override the default order.

Concatenating Strings

The + operator creates and concatenates strings:

```
String name = "Jane ";
String lastName = "Hathaway";
String fullName;
name = name + lastName;     // name is now
                            //"Jane Hathaway"
                            //    OR
name += lastName;           // same result
fullName = name;
```

Summary

In this lesson, you should have learned the following:

- Java has eight primitive data types.
- A variable must be declared before it can be used.
- Java provides a comprehensive set of operators.
- Explicit casting may be necessary if you use data types smaller than `int`.
- The `+` and `+=` operators can be used to create and concatenate strings.

Practice : Overview

This practice covers the following topics:

- Declaring and initializing variables
- Using various operators to compute new values
- Displaying results on the console