

Java Review

Java

- Java Language Overview
- Representing Data in Java
 - primitive variables
 - reference variables
- Java Program Statements
 - Conditional statements
 - Repetition statements (loops)
- Writing Classes in Java
 - Class definitions
- Arrays

Some Salient Characteristics of Java

- Java is ***platform independent***: the same program can run on any correctly implemented Java system
- Java is ***object-oriented***:
 - Structured in terms of ***classes***, which group data with operations on that data
 - Can construct new classes by ***extending*** existing ones
- Java designed as
 - A ***core language*** plus
 - A rich collection of ***commonly available packages***
- Java can be embedded in Web pages

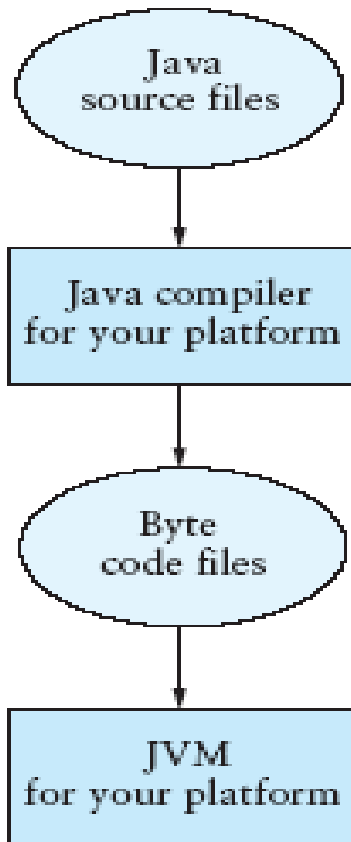
Java Processing and Execution

- Begin with Java **source code** in text files: **Model.java**
- A Java source code compiler produces Java **byte code**
 - Outputs one file per class: **Model.class**
 - May be standalone or part of an IDE
- A **Java Virtual Machine** loads and executes class files
 - May compile them to native code (e.g., x86) internally

Compiling and Executing a Java Program

FIGURE A.1

Compiling and Executing a Java Program



Introduction to Objects

- An *object* represents something with which we can interact in a program
- An object provides a collection of services that we can tell it to perform for us
- The services are defined by methods in a *class* that defines the object
- A class represents a concept, and an object represents the embodiment of a class
- A class can be used to create multiple objects

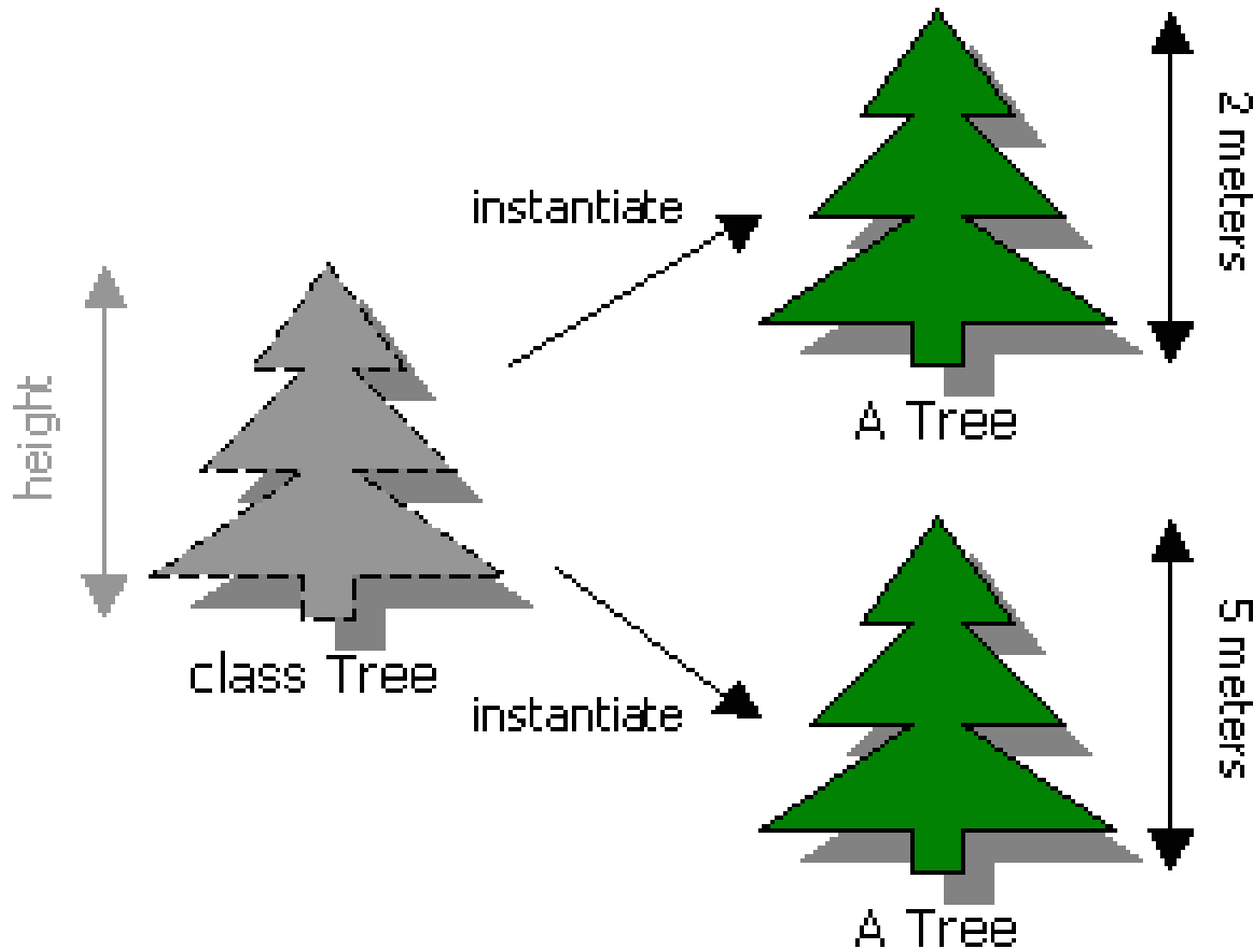


Fig. 1: Instantiating two Trees from the Tree class

Java Program Structure

- In the Java programming language:
 - A program is made up of one or more *classes*
 - A class contains one or more *methods*
 - A method contains program *statements*
- Attributes/properties correspond to fields (or variables)
- Behaviors/operations correspond to methods
- A Java application always contains a method called `main`

Java Program Structure

```
// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
    }
}
```

class body {

class header → `public class MyProgram`

method header → `public static void main (String[] args)`

method body {

Java

- Java Language Overview
- Representing Data in Java
 - primitive variables
 - reference variables
- Java Program Statements
 - Conditional statements
 - Repetition statements (loops)
- Writing Classes in Java
 - Class definitions
- Arrays

Representing Data in Java

- Variables
 - primitive – hold numbers, letters
 - reference – refer to objects (String, Tree)

Variables (Primitive or Reference)

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold

data type variable name

```
int total;
```

```
int count, temp, result;
```

Multiple variables can be created in one declaration

Primitive Data

- There are exactly eight primitive data types in Java
- Four of them represent integers:
 - `byte, short, int, long`
- Two of them represent floating point numbers:
 - `float, double`
- One of them represents characters:
 - `char`
- And one of them represents boolean values:
 - `boolean`

Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

<u>Type</u>	<u>Storage</u>	<u>Min Value</u>	<u>Max Value</u>
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	+/- 3.4×10^{38} with 7 significant digits	
double	64 bits	+/- 1.7×10^{308} with 15 significant digits	

Arithmetic Expressions

- An *expression* is a combination of one or more operands and their operators

- *Arithmetic expressions* use the

operators:

Addition

+

Subtraction

-

Multiplication

Division

/

Remainder

%

(no ^ operator)

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Division and Remainder

- If both operands to the division operator ($/$) are integers, the result is an integer (the fractional part is discarded)

14 / 3 equals? 4

8 / 12 equals? 0

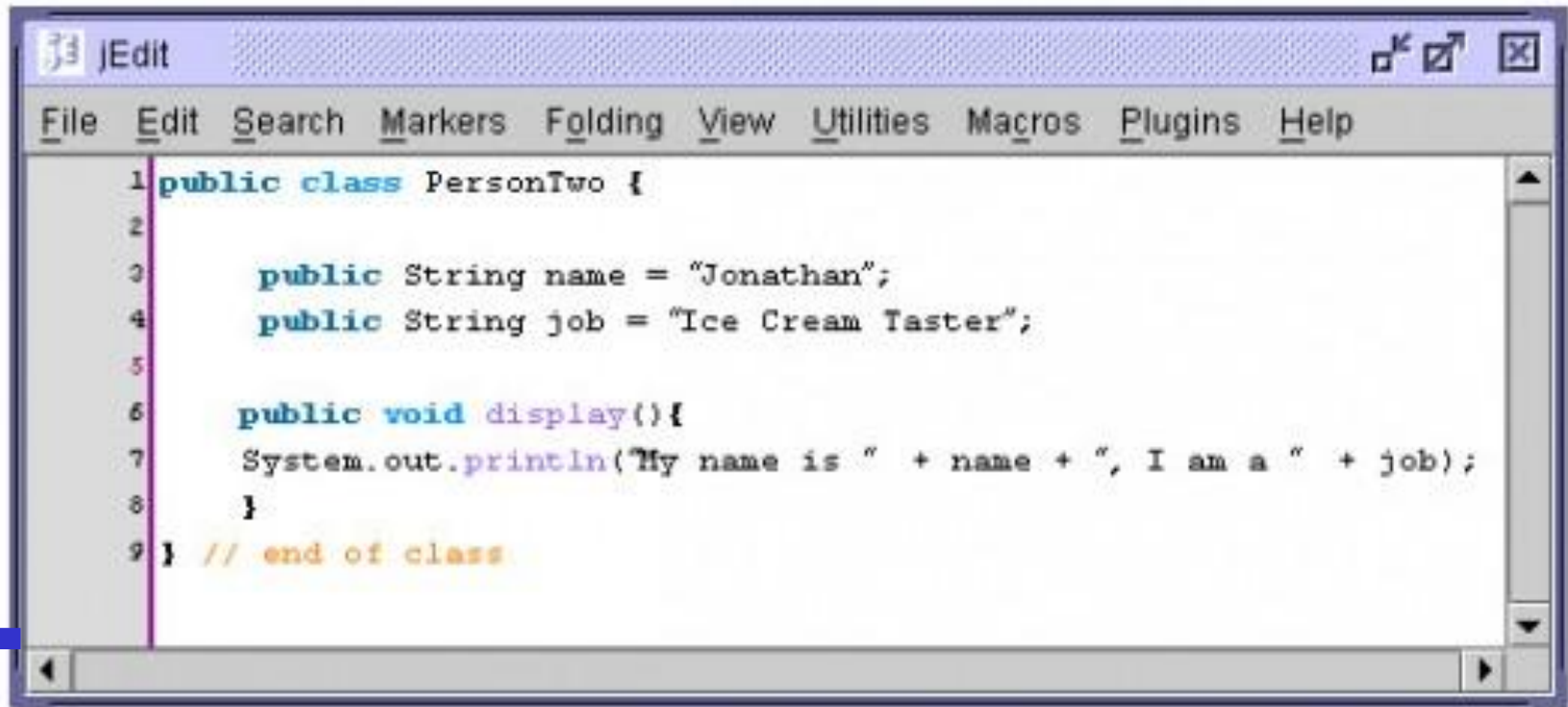
- The remainder operator ($\%$) returns the remainder after dividing the second operand into the first

14 % 3 equals? 2

8 % 12 equals? 8

String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another
- The plus operator (+) is also used for arithmetic addition
- The function that the + operator performs depends on the type of the information on which it operates
 - If at least one operand is a string, it performs string concatenation
 - If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order



```
1 public class PersonTwo {
2
3     public String name = "Jonathan";
4     public String job = "Ice Cream Taster";
5
6     public void display(){
7         System.out.println("My name is " + name + ", I am a " + job);
8     }
9 } // end of class
```

Explain why the value of the expression

`2 + 3 + "test"` is `"5test"`

while the value of the expression

`"test" + 2 + 3` is `"test23"`

What is the value of `"test" + 2 * 3` ?

Data Conversions

- In Java, data conversions can occur in three ways:
 - assignment conversion
 - arithmetic promotion
 - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
 - Only widening conversions can happen via assignment
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

Data Conversions

- *Casting* is the most powerful, and dangerous, technique for conversion
 - Both widening and narrowing conversions can be accomplished by explicitly casting a value
 - To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (float) total / count;
```



- This is a technical joke about the Java programming language.
- In Java, if a piece of data isn't of the type that some bit of program would normally accept, you have to "cast" it to the expected type. You do this by writing the name of the expected type, in parens, in front of the piece of data.

Reference Variables

- A variable holds either a primitive type or a *reference* to an object
- A class name can be used as a type to declare an *object reference variable*

```
String title;
```

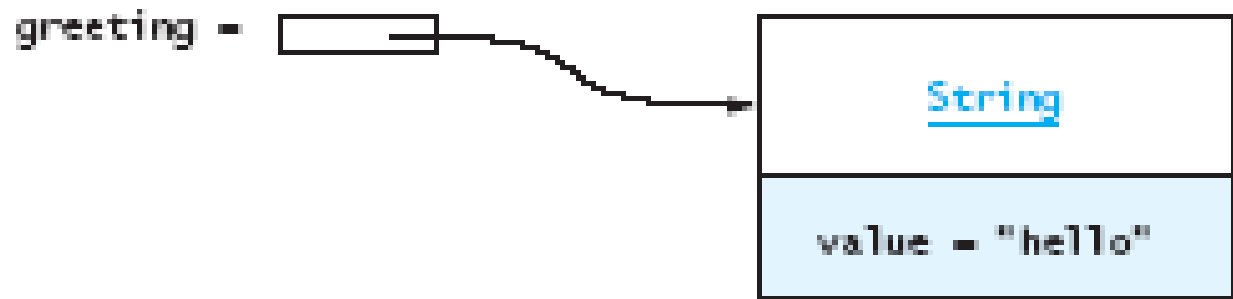
- No object is created with this declaration
- An object reference variable holds the address of an object
- The object itself must be created separately

Referencing and Creating Objects

- When you **declare reference variables**
 - They reference objects of **specified types**
- Two reference variables can reference **the same object**
- The **new** operator creates an instance of a class
- A **constructor** executes when a new object is created
- Example: `String greeting = "hello";`

FIGURE A.2

Variable `greeting`
References a `String`
Object



Creating Objects

- Generally, we use the `new` operator to create an object

```
title = new String ("Java Software Solutions");
```



This calls the `String` *constructor*, which is a special method that sets up the object

- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

Java

- Java Language Overview
- Representing Data in Java
 - primitive variables
 - reference variables
- **Java Program Statements**
 - Conditional statements
 - Repetition statements (loops)
- Writing Classes in Java
 - Class definitions
- Arrays

Java Control Statements:

- Branches

- if
- if-else
- switch

- Loops

- while
- do-while
- for

Java Control Statements:

Rules of Thumb

- *Learn program patterns* of general utility (branching, loops, etc.) and *use relevant patterns* for the problem at hand
- *Seek inspiration* by systematically working test data by hand and ask yourself: “what am I doing?”
- *Declare variables* for each piece of information you maintain when working problem by hand
- *Decompose* problem into manageable tasks
- *Remember* the problem’s boundary conditions
- *Validate* your program by tracing it on test data with known output

Branches/Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next
- Therefore they are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- Java's conditional statements are
 - the *if statement*
 - the *if-else statement*
 - the *switch statement*

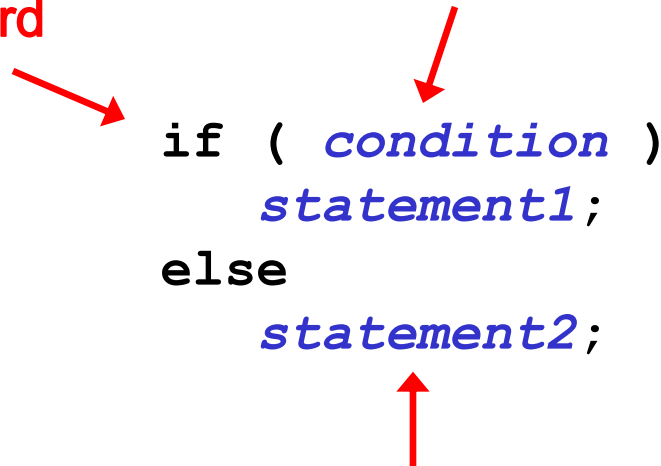
The if Statement

- The *if statement* has the following syntax:

`if` is a Java reserved word

The *condition* must be a boolean expression. It must evaluate to either true or false.

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



If the *condition* is true, *statement1* is executed.
If it is false, *statement2* is executed.

Boolean Expressions

- A condition often uses one of Java's *equality operators* or *relational operators*, which all return boolean results:

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

- Note the difference between the equality operator (==) and the assignment operator (=)

Logical Operators

- Boolean expressions can use the following *logical operators*:

! Logical NOT

& & Logical AND

| | Logical OR

- They all take boolean operands and produce boolean results
- Logical NOT is a unary operator (it operates on one operand)
- Logical AND and logical OR are binary operators (each operates on two operands)

Example if statement

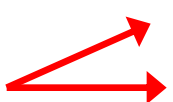
```
public static void main(String[] args) {  
  
    int user = 21;  
  
    if (user <= 18) {  
        System.out.println("User is 18 or younger");  
    }  
    else if (user > 18 && user < 40) {  
        System.out.println("User is between 19 and 39");  
    }  
  
    else {  
        System.out.println("User is older than 40");  
    }  
}
```


The switch Statement


- The general syntax of a `switch` statement is:

```
switch ( expression )
{
    case value1 :
        statement-list1;
        break;
    case value2 :
        statement-list2;
        break;
    case value3 :
        statement-list3;
        break;
    case ...
}
```

`switch`
`and`
`case`
`are`
`reserved`
`words`



←
If *expression*
matches *value2*,
control jumps
to here



Example Switch statement

```
public static void main(String[] args) {  
  
    int user = 18;  
  
    switch ( user ) {  
        case 18:  
            System.out.println("You're 18");  
            break;  
        case 19:  
            System.out.println("You're 19");  
            break;  
        case 20:  
            System.out.println("You're 20");  
            break;  
        default:  
            System.out.println("You're not 18, 19 or 20");  
    }  
  
}
```

LOOPS: Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
 - the *while loop*
 - the *do loop*
 - the *for loop*
- The programmer should choose the right kind of loop for the situation

The while Statement

- The *while statement* has the following syntax:

`while` is a reserved word → `while (condition)
statement;`

If the *condition* is true, the *statement* is executed.
Then the *condition* is evaluated again.

The *statement* is executed repeatedly until
the *condition* becomes false.

Example

```
//*****
// Counter.java      Author: Lewis/Loftus
//
// Demonstrates the use of a while loop.
//*****

public class Counter
{
    //-----
    // Prints integer values from 1 to a specific limit.
    //-----
    public static void main (String[] args)
    {
        final int LIMIT = 5;
        int count = 1;

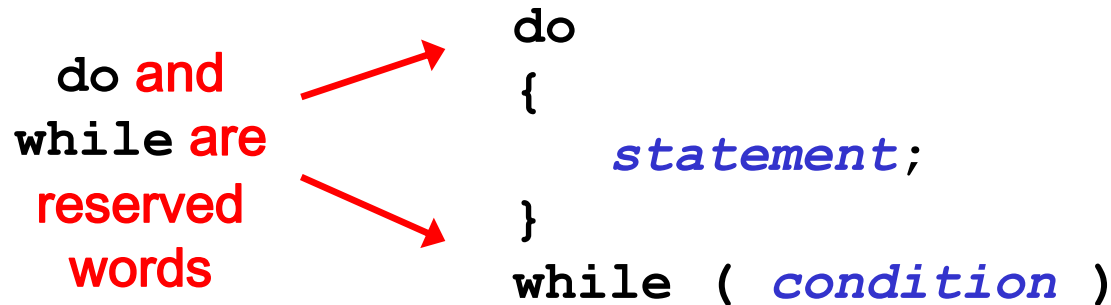
        while (count <= LIMIT)
        {
            System.out.println (count);
            count = count + 1;
        }

        System.out.println ("Done");
    }
}
```

The do Statement

- The *do statement* has the following syntax:

do and
while are
reserved
words



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

The *statement* is executed once initially,
and then the *condition* is evaluated

The *statement* is executed repeatedly
until the *condition* becomes false

The do Statement

- A `do` loop is similar to a `while` loop, except that the condition is evaluated after the body of the loop is executed
- Therefore the body of a `do` loop will execute at least once

Example

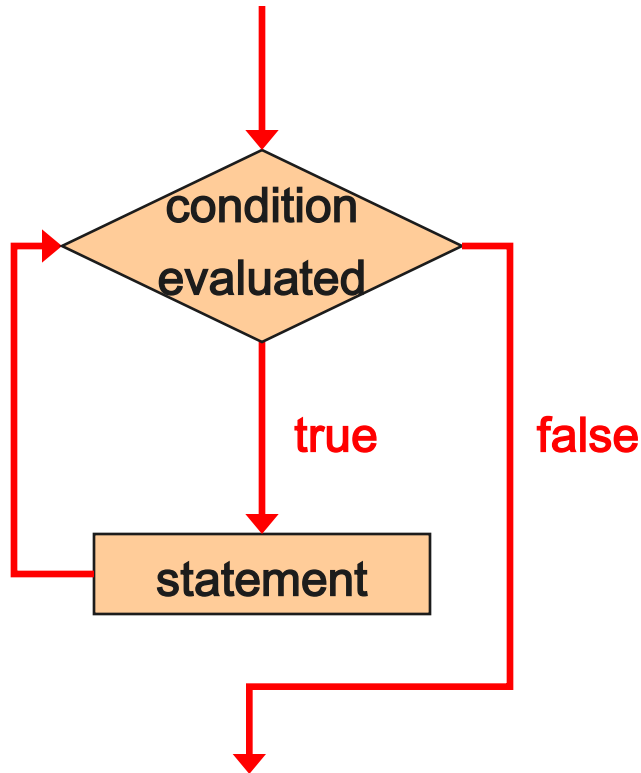
```
/**
 * Counter2.java      Author: Lewis/Loftus
 *
 * Demonstrates the use of a do loop.
 */
public class Counter2
{
    //-----
    // Prints integer values from 1 to a specific limit.
    //-----
    public static void main (String[] args)
    {
        final int LIMIT = 5;
        int count = 0;

        do
        {
            count = count + 1;
            System.out.println (count);
        }
        while (count < LIMIT);

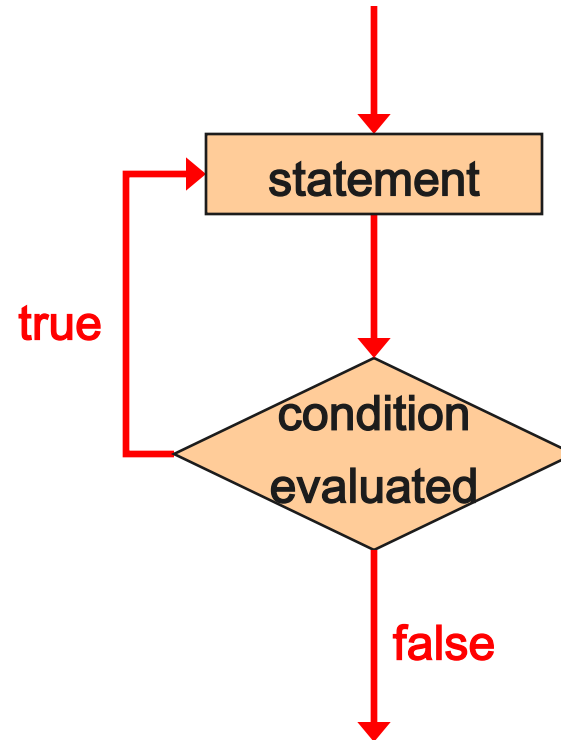
        System.out.println ("Done");
    }
}
```


Comparing while and do

while loop

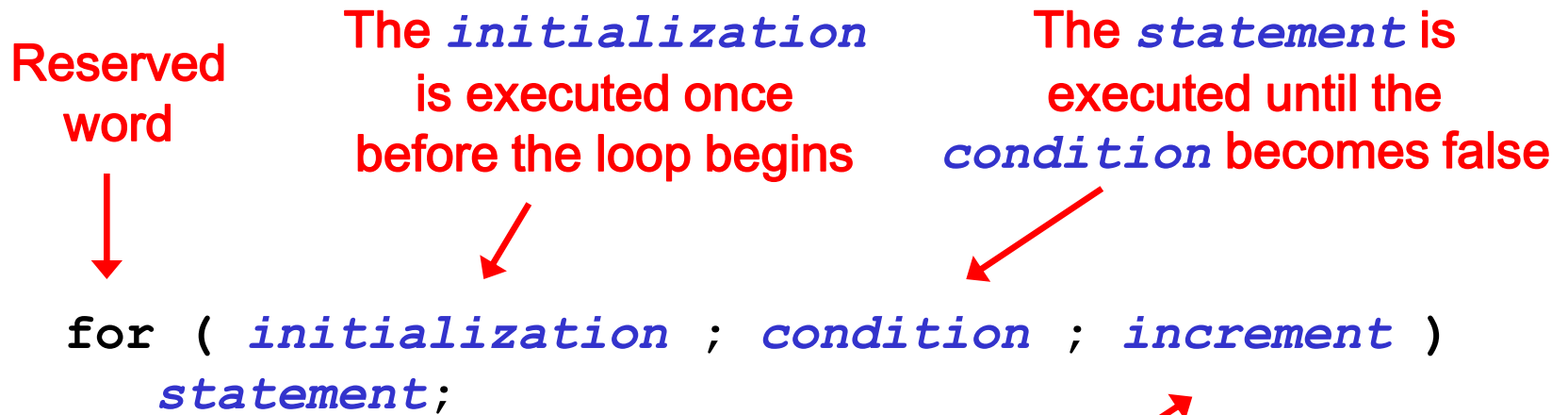


do loop



The for Statement

- The *for statement* has the following syntax:



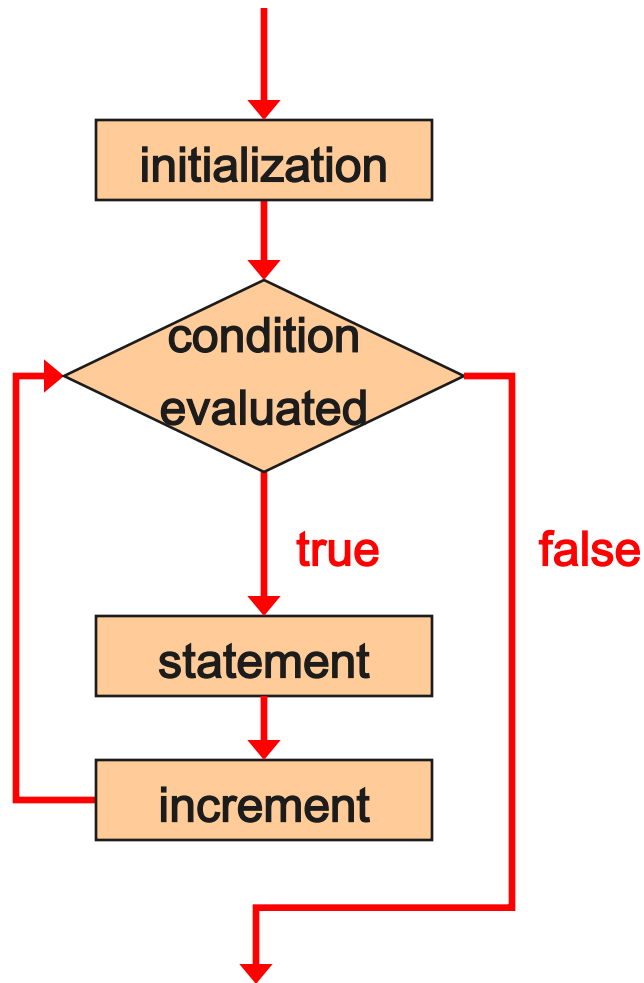
The *increment* portion is executed at the end of each iteration
The *condition-statement-increment* cycle is executed repeatedly

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;  
while ( condition )  
{  
    statement;  
    increment;  
}
```

Logic of a for loop



The for Statement

- Like a `while` loop, the condition of a `for` statement is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times
- It is well suited for executing a loop a specific number of times that can be determined in advance

Example

```
/** *****  
// Counter3.java          Author: Lewis/Loftus  
//  
// Demonstrates the use of a for loop.  
// *****  
  
public class Counter3  
{  
    //-----  
    // Prints integer values from 1 to a specific limit.  
    //-----  
    public static void main (String[] args)  
    {  
        final int LIMIT = 5;  
  
        for (int count=1; count <= LIMIT; count++)  
            System.out.println (count);  
  
        System.out.println ("Done");  
    }  
}
```

Choosing a Loop Structure

- When you can't determine how many times you want to execute the loop body, use a `while` statement or a `do` statement
 - If it might be zero or more times, use a `while` statement
 - If it will be at least once, use a `do` statement
- If you can determine how many times you want to execute the loop body, use a `for` statement

Java

- Java Language Overview
- Representing Data in Java
 - primitive variables
 - reference variables
- Java Program Statements
 - Conditional statements
 - Repetition statements (loops)
- Writing Classes in Java
 - Class definitions
- Arrays

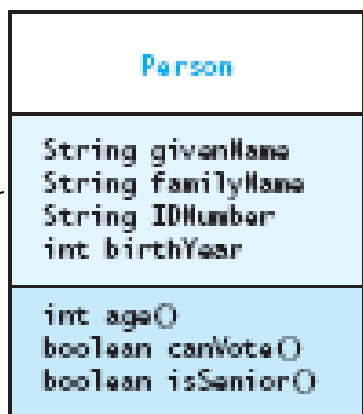
Objects and Classes

- An **object** has:
 - *state* - descriptive characteristics
 - *behaviors* - what it can do (or what can be done to it)
- A **class** is the model or pattern from which objects are created
- For example, consider a coin that can be flipped so that its face shows either "heads" or "tails"
- The state of the coin is its current face (heads or tails)
- The behavior of the coin is that it can be flipped

Defining Your Own Classes

- *Unified Modeling Language (UML)* is a standard diagram notation for describing a class

FIGURE A.6
Class Diagram for
Person

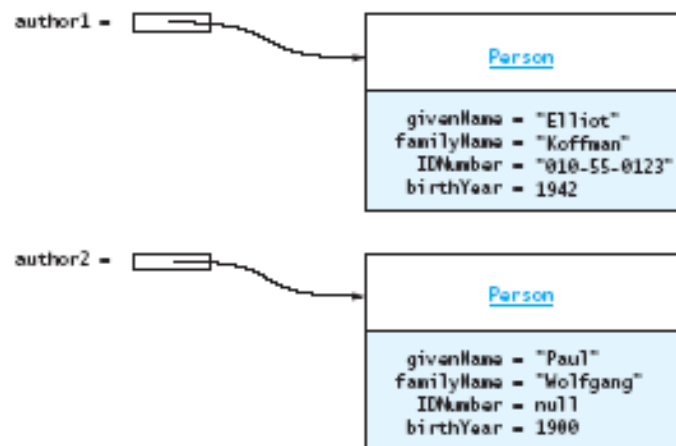


Field
signatures:
type and name

Method *signatures:*
name, argument
types, result type

Class
name

FIGURE A.7
Object Diagrams of
Two Instances of Class
Person



Field
values

Class
name

Defining Your Own Classes (continued)

- The modifier `private` limits access to just this class
- Only class members with `public` visibility can be accessed outside of the class* (* but see `protected`)
- **Constructors** initialize the data fields of an instance

TABLE A.11

Default Values for Data Fields

Data Field Type	Default Value
<code>int</code> (or other integer type)	<code>0</code>
<code>double</code> (or other real type)	<code>0.0</code>
<code>boolean</code>	<code>false</code>
<code>char</code>	<code>\u0000</code> (the smallest Unicode character: the null character)
Any reference type	<code>null</code>

The Person Class

```
// we have omitted javadoc to save space
public class Person {
    private String givenName;
    private String familyName;
    private String IDNumber;
    private int birthYear;

    private static final int VOTE_AGE = 18;
    private static final int SENIOR_AGE = 65;
    ...
}
```

The Person Class (2)

```
// constructors: fill in new objects
public Person(String first, String family,
               String ID, int birth) {
    this.givenName    = first;
    this.familyName   = family;
    this.IDNumber     = ID;
    this.birthYear    = birth;
}
public Person (String ID) {
    this.IDNumber = ID;
}
```

The Person Class (3)

```
// modifier and accessor for givenName
public void setGivenName (String given) {
    this.givenName = given;
}

public String getGivenName () {
    return this.givenName;
}
```

The Person Class (4)

```
// more interesting methods ...
public int age (int inYear) {
    return inYear - birthYear;
}
public boolean canVote (int inYear) {
    int theAge = age(inYear);
    return theAge >= VOTE_AGE;
}
```

The Person Class (5)

```
// "printing" a Person
public String toString () {
    return "Given name: " + givenName + "\n"
        + "Family name: " + familyName + "\n"
        + "ID number: " + IDNumber + "\n"
        + "Year of birth: " + birthYear + "\n";
}
```


The Person Class (6)

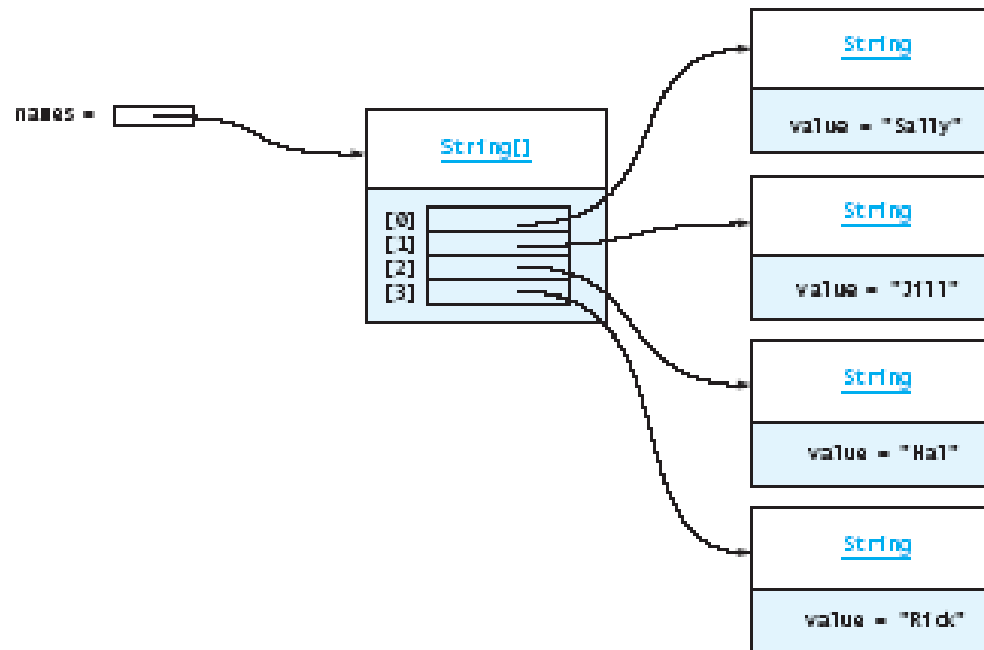
```
// same Person?  
public boolean equals (Person per) {  
    return (per == null) ? false :  
        this.IDNumber.equals(per.IDNumber) ;  
}
```

Java

- Java Language Overview
- Representing Data in Java
 - primitive variables
 - reference variables
- Java Program Statements
 - Conditional statements
 - Repetition statements (loops)
- Writing Classes in Java
 - Class definitions
- Arrays

Arrays

- In Java, an array is also an object
- The elements are indexes and are referenced using the form **arrayvar[subscript]**



Array Example

```
float grades[] = new float[numStudents];  
... grades[student] = something; ...
```

```
float total = 0.0;  
for (int i = 0; i < grades.length; i++) {  
    total += grades[i];  
}
```

```
System.out.println("Average =" +  
                    total / numStudents);
```

Array Example Variations

```
// visit cells in reverse order
for (int i = grades.length-1; i >= 0; i-- )
    {
        total += grades[i];
    }
```

```
// uses Java 5.0 "for each" looping
for (float grade : grades) {
    total += grade;
}
```