Advanced Object-Oriented Programming

Introduction to OOP and Java

Dr. Kulwadee Somboonviwat

International College, KMITL

kskulwad@kmitl.ac.th

Course Objectives

- Solidify **object-oriented programming** skills
- Study the Java Technology
  - The Java Programming Language
  - The Java Platform, Enterprise Edition (Java EE 7)

Key Topics covered in this course

- Fundamentals of Java Programming
- Object-oriented programming concepts
- GUI Programming
- Concurrency
- Java EE 7

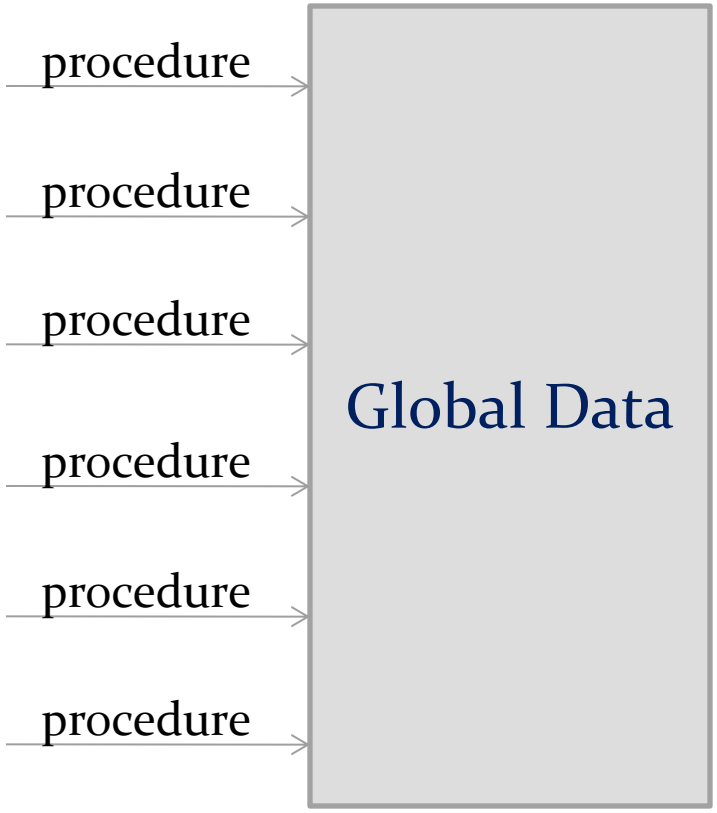
Object-Oriented Programming

- Dominant programming paradigm these days
- A program is made of **objects**.
- Each object
  - exposes specific functionality to the users
  - encapsulates (hides) the implementation of its functionality

Traditional Procedural Programming

- 1970s: "**structured**", procedural programming
  - **Programs = Algorithms + Data** (Niklaus Wirth, 1975)
    - First, we think about a set of procedures (algorithms) needed to solve our problem.
    - Then, we find appropriate ways to store the data
  - Used in C, Pascal, Basic, etc.
  - Structured programming works well for small to medium sized problems

In procedural programming,
- problem is decomposed into **procedures**
- all procedures manipulate a set of **global data**

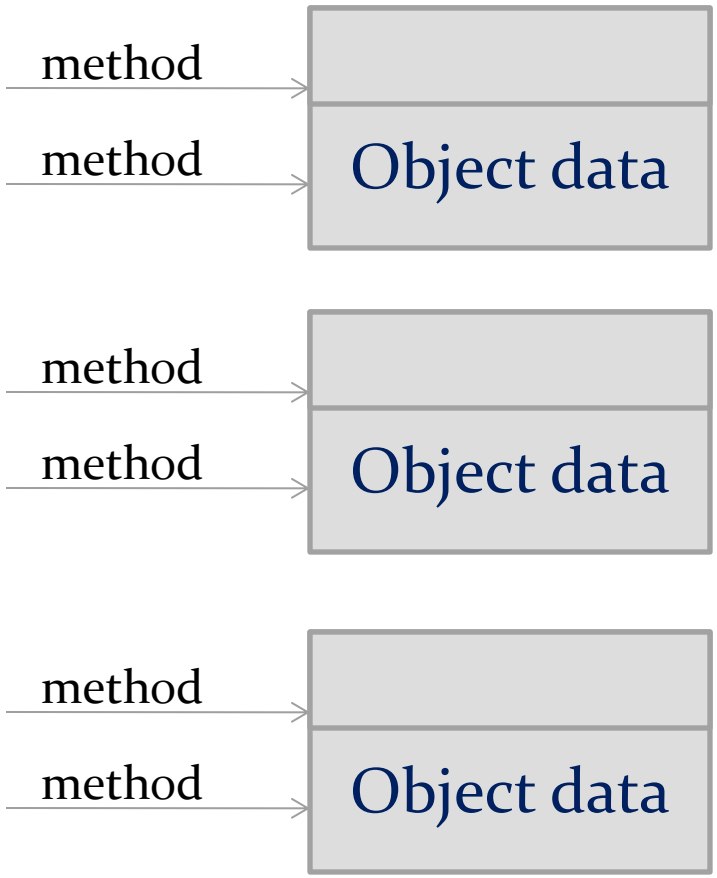


Suppose that …
- your program has 2,000 procedures
- a piece of data is in an incorrect state

How are you going to find bugs in this situation?

How many procedures you need to search for the culprit?

In object-oriented programming style,
- your program consists of **objects**
- each object has a specific set of **attributes** and **methods**



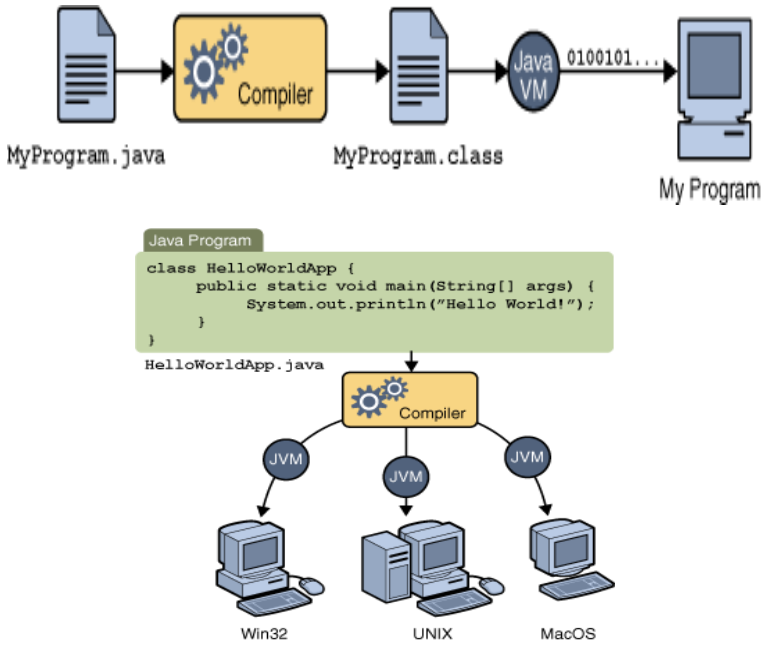
Suppose that …
- your program has 200 objects, and each object has 10 methods.
- a piece of data ***of an object*** is in an incorrect state

How are you going to find bugs in this situation?

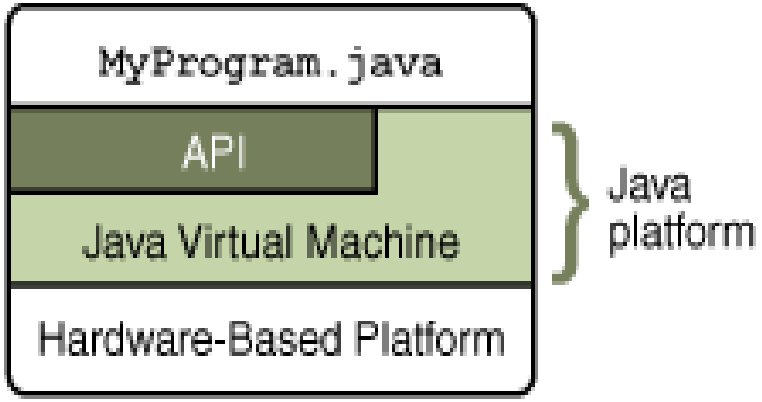
How many procedures you need to search for the culprit?

Java Technology

The Java Programming Language



The Java Platform



Source: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Characteristics of the Java PL

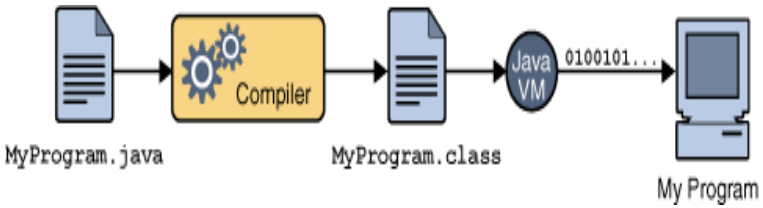
- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High Performance
- Robust
- Secure

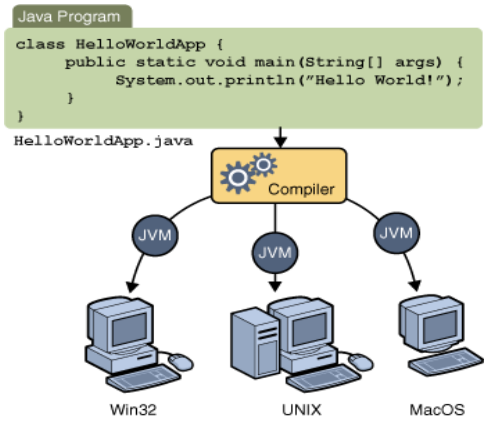
C++ versus Java

Features	Java	C++
Data types	Supports both primitive scalar types and classes	Supports both primitive scalar types and classes
Object allocation	Allocated from **heap**, accessed through reference variables (no pointers)	Allocated from **heap** or **stack**, accessed through reference variables or **pointers**
Object de-allocation	Implicit (garbage collection)	Explicit (delete operator)
Inheritance	Single inheritance only (multiple inheritance is possible with **interfaces**)	Single, Multiple inheritance
Binding	All binding of messages to methods are dynamic except in the case of methods that cannot be overridden	Dynamic binding of messages to methods are **optional (using the virtual keyword)**

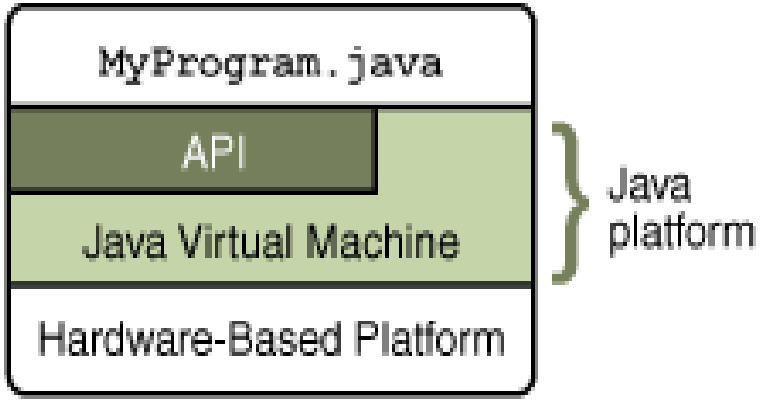
Java Technology

The Java Programming Language



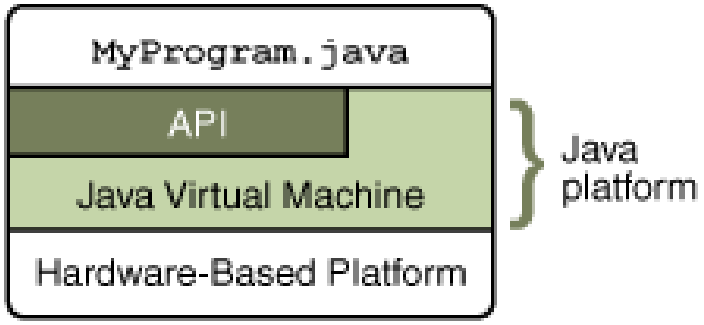


The Java Platform



Source: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java As a Programming Platform



- A *platform* is the hardware or software environment in which a program runs.
  - E.g. Windows, Linux, Solaris OS, and Mac OS
- Java is a software-only platform that runs on top of other hardware-based platforms. It consists of
  - **The Java Virtual Machine**: a software-based processor that presents its own instruction set
  - **The Java Application Programming Interface (API)**

Source: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

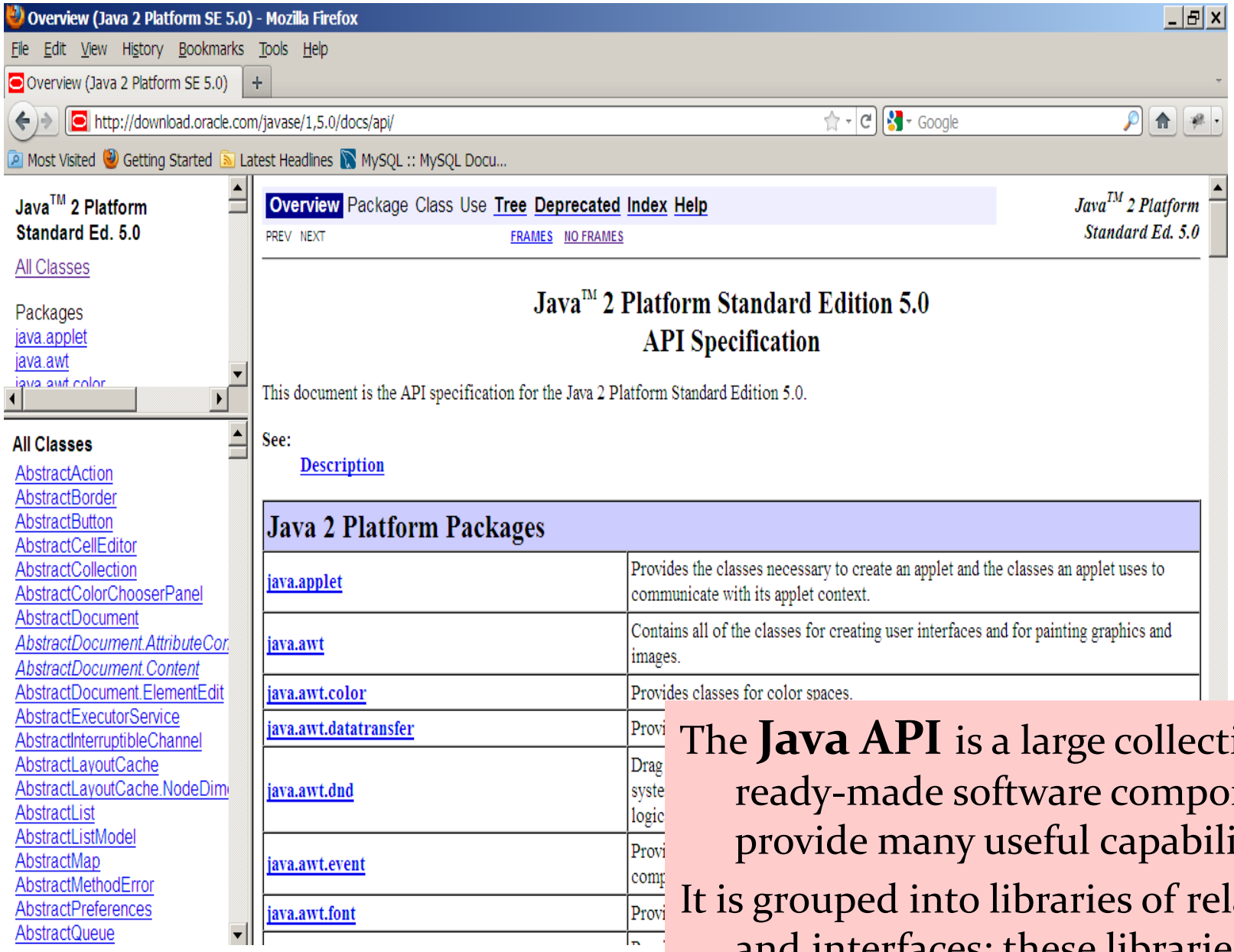
Different Editions of the Java Platform

- ***Java Platform, Standard Edition (Java SE):***
  - *stand-alone programs that run on* desktops.
  - *applets (*programs that run in the context of a web browser)
- ***Java Platform, Enterprise Edition (Java EE):***
  - built on top of Java SE.
  - enterprise-oriented applications and *servlets (*server programs that conform to Java EE's Servlet API).
- ***Java Platform, Micro Edition (Java ME):***
  - *MIDlets (programs that run on mobile* information devices)
  - *Xlets (which are programs that run on* embedded devices)

Java Jargon

Table 2–1 Java Jargon

Name	Acronym	Explanation
Java Development Kit	JDK	The software for programmers who want to write Java programs
Java Runtime Environment	JRE	The software for consumers who want to run Java programs
Standard Edition	SE	The Java platform for use on desktops and simple server applications
Enterprise Edition	EE	The Java platform for complex server applications
Micro Edition	ME	The Java platform for use on cell phones and other small devices
Java 2	J2	An outdated term that described Java versions from 1998 until 2006
Software Development Kit	SDK	An outdated term that described the JDK from 1998 until 2006
Update	u	Sun's term for a bug fix release
NetBeans	—	Sun's integrated development environment

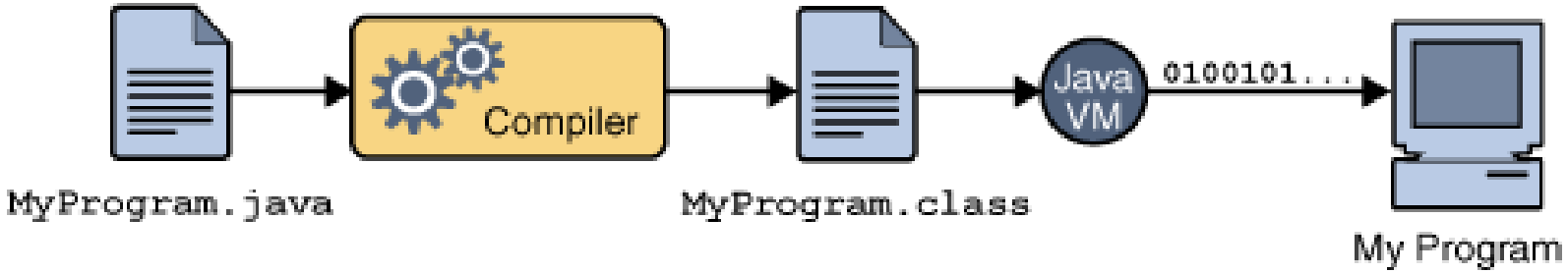


The **Java API** is a large collection of ready-made software components that provide many useful capabilities.

It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*

Source: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Software Development Process



- Write the source code and save in files with **.java** extension
- Compile the source code into **.class** files using the **javac compiler**
- A **.class** file contains **bytecodes**

(the machine language of the Java Virtual Machine (Java VM)

- Run the application (with an instance of the Java VM)

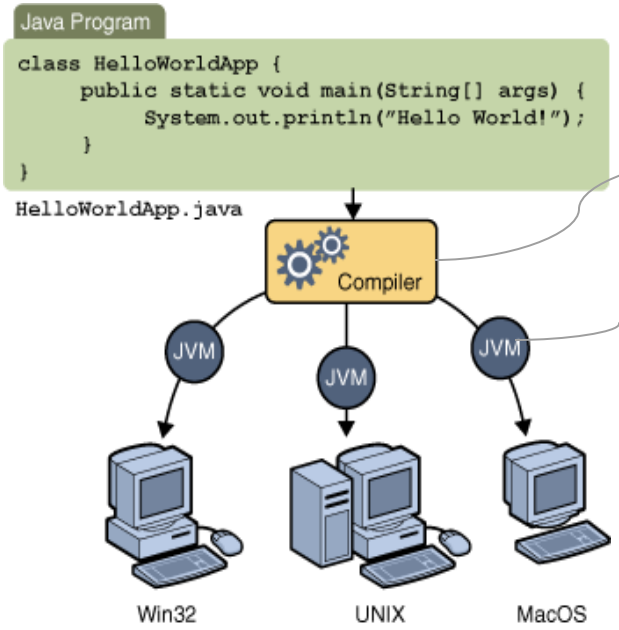
using the **java** launcher tool.

Source: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Program Execution

- The **java** tool loads and starts the VM, and passes the program's main classfile (.class) to the machine
- The VM uses **classloader** to load the classfile
- The VM's **bytecode verifier** checks that the classfile's bytecode is valid and does not compromise security
  - If the bytecode has any problem, the verifier terminates the VM
- If all is well with the bytecode, *the VM's **interpreter** interprets the bytecode one instruction at a time*

* *Interpretation consists of identifying* bytecode instructions , and executing equivalent native instructions (instructions understood by the physical processor)



$ javac HelloWorldApp.java

$ java HelloWorldApp
(1)Load the JVM
(2)classloader loads HelloWorldApp.class
(3)bytecode verifier check that the classfile is valid and secure
(4)If all is well, the **interpreter** interpret the bytecode
(5)A section of frequently executed bytecode will be compiled to native code by the **JIT (Just In Time) compiler**

- The Java platform provides an abstraction over the underlying hardware/OS platform
  - **Portability**: the same .class files can run unchanged on a variety of hardware platforms and operating systems

What can Java Technology Do?

- Development Tools: javac, java, javadoc
- Rich APIs
- Deployment Technologies: Web Start, Java Plug-In
- User Interface Toolkits
- Integration libraries: JDBC, JNDI, RMI

Java Basics

- (simple) Program Structure
- Comments

A Simple Java Program

```
/**
 * File: FirstSample.java
 * This is our first sample program in Java
 * @version 1
 * @author Kulwadee
 */

public class FirstSample
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to Java!");
    }
}
```

class keyword : everything in java program must be inside a class!

class name: starts with a letter, followed by any number of letters or digits

Access modifier

The main method: the method that every java program MUST have!

A Simple Java Program : output a line of message to console

```
System.out.println( "Welcome to Java! ");
```

```
Object.method(parameters)
```

Now.. Let's compile and run our first program!

C:\> **javac FirstSample.java**

C:\> **dir FirstSample.***

FirstSample.java FirstSample.class

C:\> **java FirstSample**

Welcome to Java!

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types

Primitive Data Types (1/3)

Type	Description	Size
int	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
byte	The type describing a single byte, with range -128 . . . 127	1 byte
short	The short integer type, with range -32768 . . . 32767	2 bytes
long	The long integer type, with range -9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807	8 bytes

Primitive Data Types (2/3)

Type	Description	Size
double	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
float	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes

Primitive Data Types (3/3)

Type	Description	Size
char	The character type, representing code units in the **Unicode encoding scheme**	2 bytes
boolean	The type with the two truth values false and true	1 bit

Java is a statically, strongly typed language

- ***Statically typed*** *:*

every variable must be declared with a data type.

(vs. *dynamically typed*)

- ***Strongly typed*** *:*

JVM keeps track of all variable types. Once a variable is declared, its data type cannot be changed.

(vs. *weakly typed)*

What about these languages?

C++

C

Python

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types
- Declaring Variables

Types and Variables

Syntax 2.1: Variable Definition

typeName variableName = value;

or

typeName variableName;

Example:

```
String greeting = "Hello, AOOP!";
double salary = 65000.0;
```

Purpose:

To define a new variable of a particular type and optionally supply an initial value

Identifiers

- Identifier: **name of a variable, method, or class**
- Rules for identifiers in Java:
  - Can be made up of letters, digits, and the underscore (_) character
  - Cannot start with a digit
  - Cannot use other symbols such as ? or %
  - Spaces are not permitted inside identifiers
  - You cannot use reserved words
  - They are **case sensitive**
- **Convention:**
  - variable names start with a lowercase letter
  - class names start with an uppercase letter

Number Types

- **int**: integers, no fractional part

  1, -4, 0

- **double**: floating-point numbers (double precision)

  0.5, -3.11111, 4.3E24, 1E-14

- A numeric computation *overflows* if the result falls outside the range for the number type

```
int n = 1000000;
System.out.println(n * n); // prints -727379968
```

Number Types: Floating-point

- Rounding errors occur when an exact conversion between numbers is not possible

```
double f = 4.35;
System.out.println(100 * f); // prints 434.99999999999994
```

- Java: Illegal to assign a floating-point expression to an integer variable

```
double balance = 13.75;
int dollars = balance; // Error
```

- **Casts: used to convert a value to a different type**

```
int dollars = (int) balance; // OK
```

  - Cast discards fractional part.
  - Math.round converts a floating-point number to nearest integer

```
long rounded = Math.round(balance);
// if balance is 13.75, then rounded is set to 14
```

Cast

Cast: used to convert a value to a different type
→ discard fractional part

Syntax 2.2: Cast

(typeName) expression

Example:

(int) (balance * 100)

Purpose:

To convert an expression to a different type

Constants: final

- A final variable is a constant
- Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: use all-uppercase names for constants

```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;

payment = dollars + quarters * QUARTER_VALUE +
                    dimes * DIME_VALUE +
                    nickels * NICKEL_VALUE +
                    pennies * PENNY_VALUE;
```

Constants: static final

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as static and final
- Give static final constants public access to enable other classes to use them

```
public class Math
{
. . .
public static final double E = 2.71828182845904523536;
public static final double PI = 3.14159265358979323846;
}
```

```
double circumference = Math.PI * diameter;
```

Constant Definition

Syntax 2.3: Constants

In a method:
final *typeName variableName = expression* ;

In a class:
accessSpecifier static final *typeName variableName = expression*;

Example:

```
final double NICKEL_VALUE = 0.05;
public static final double LITERS_PER_GALLON = 3.785;
```

Purpose:

To define a constant in a method or a class

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types
- Declaring Variables
- Operators

Operators

- Assignment (=), Increment (++), Decrement (--)
- Arithmetic Operators

  + - * / %
- Relational Operators

  < <= > >= == !=
- Logical Operators

  ! && || ^

Assignment, Increment, Decrement

- Assignment **is not the same as mathematical equality**:

  items = items + 1;

- Increment

  items++ is the same as items = items + 1

- Decrement

  items-- subtracts 1 from items

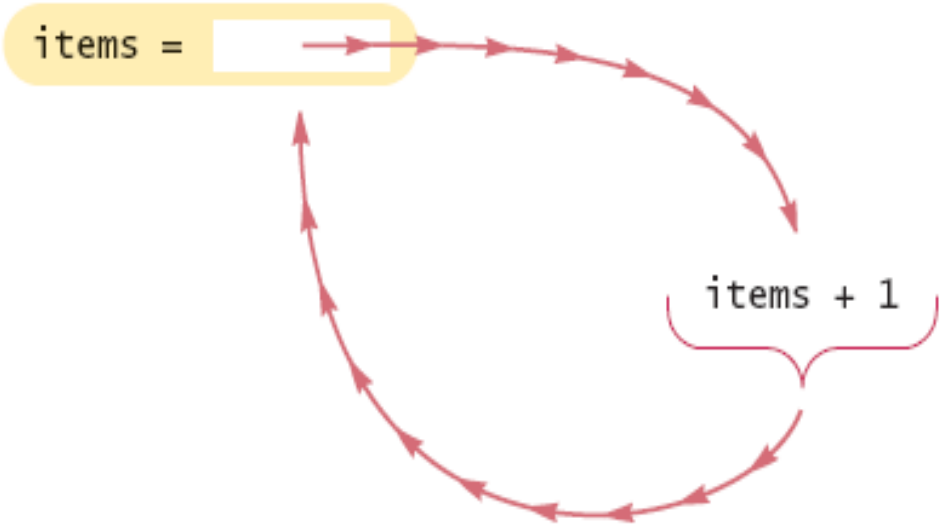


Figure 1
Incrementing a Variable

Arithmetic Operations

- / is the division operator

  If both arguments are integers, the result is an integer.

  The remainder is discarded

  7.0 / 4 yields 1.75

  7 / 4 yields 1

- Get the remainder with % (pronounced "modulo")

  7 % 4 is 3

The Math class

- Math class: contains methods like sqrt and pow
- To compute x^n, you write Math.pow(x, n)
- To take the square root of a number, use the Math.sqrt; for example, Math.sqrt(x)

- In Java,

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

can be represented as

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

The Math class

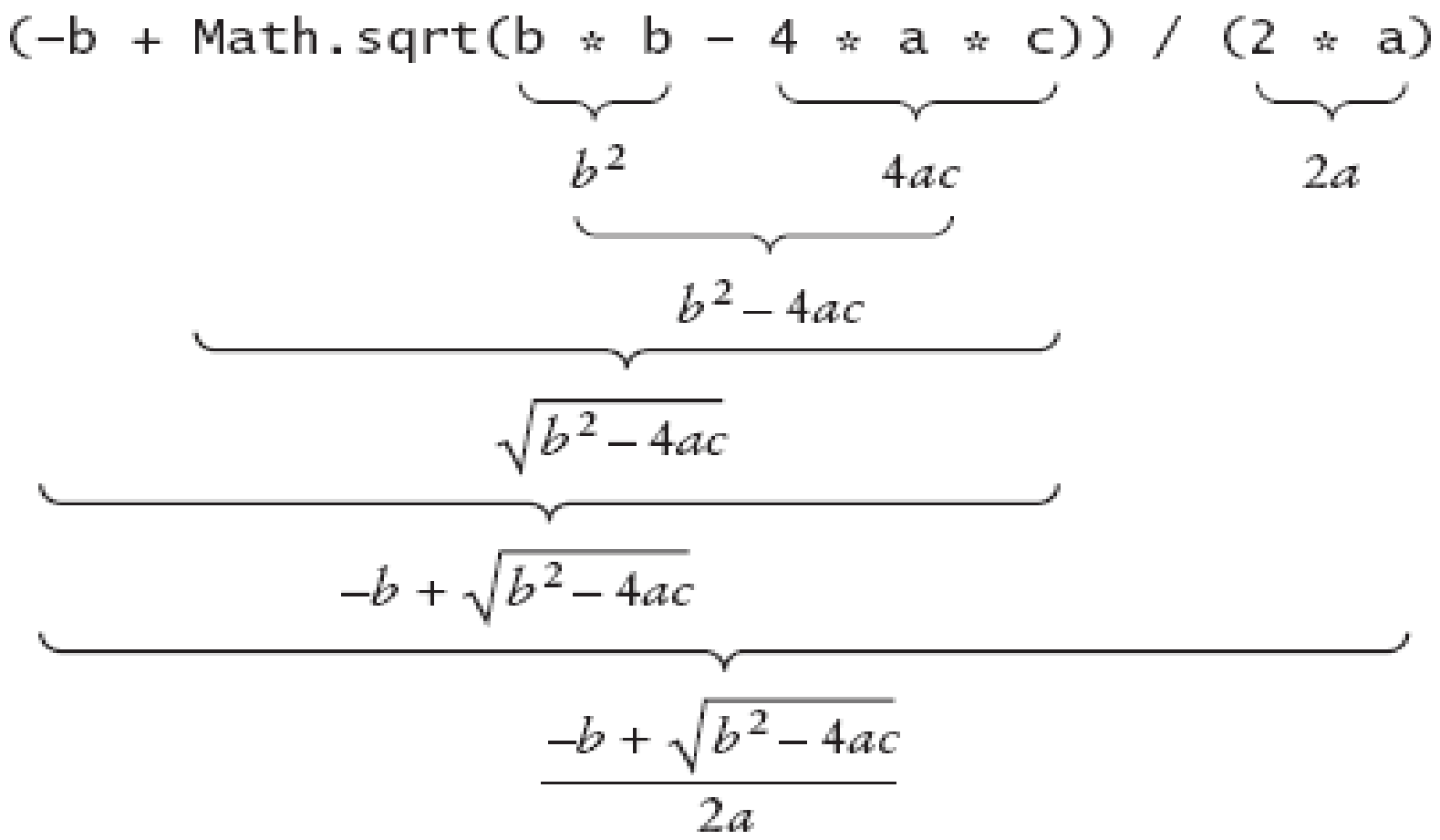


Figure 2 Analyzing an Expression

Mathematical Methods in Java

Math.sqrt(x)	square root
Math.pow(x, y)	power x^y
Math.exp(x)	e^x
Math.log(x)	natural log
Math.sin(x), Math.cos(x), Math.tan(x)	sine, cosine, tangent (x in radian)
Math.round(x)	closest integer to x
Math.min(x, y), Math.max(x, y)	minimum, maximum

Table 3–4 Operator Precedence

Operators	Associativity
[] . () (method call)	Left to right
! ~ ++ -- + (unary) - (unary) () (cast) new	Right to left
* / %	Left to right
+ -	Left to right
<< >> >>>	Left to right
< <= > >= instanceof	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &= |= ^= <<= >>= >>>=	Right to left

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types
- Declaring Variables
- Operators
- String

String

- A string is a sequence of characters
- Strings are objects of the String class
- String constants:

  "Hello, World!"
- String variables:

  String message = "Hello, World!";
- String length:

  int n = message.length();
- Empty string: ""

Figure 3 String Positions

String Operations (1)

- **Concatenation**
  - Use the + operator:String name = "Dave";

```
String message = "Hello, " + name;
// message is "Hello, Dave"
```

  - If one of the arguments of the + operator is a string, the other is converted to a stringString a = "Agent";

```
int n = 7;
String bond = a + n; // bond is Agent7
```

String Operations (2)

- **Substring**

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5);
// sub is "Hello“
```

– Supply start and “past the end” position

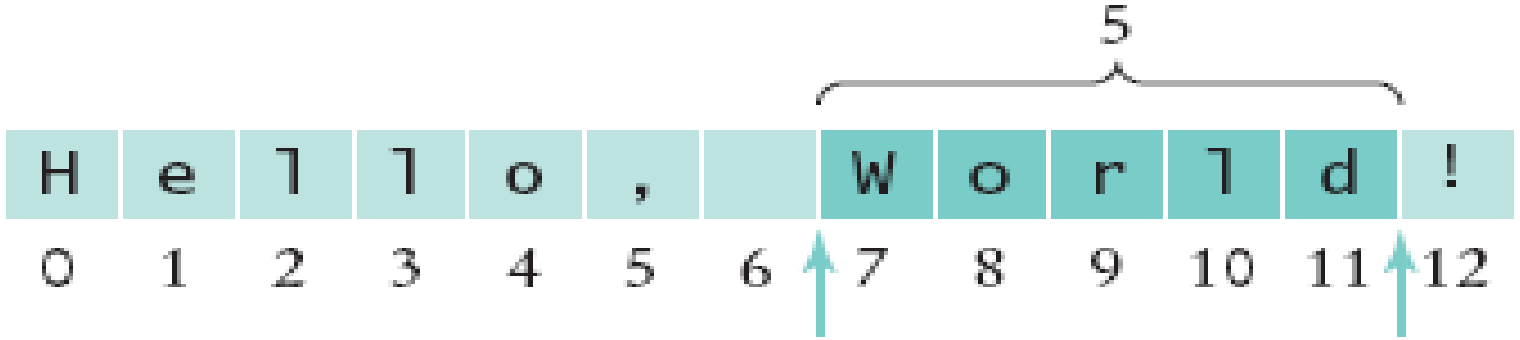


Figure 4 Extracting a Substring

String Operations (3)

- **Testing Strings for Equality**

  - use the **equals** method

    s.equals(t)

  - Do not use == to test if two strings are equal!!

  it only determines if the strings are stored in the same location or not.

```
String greeting = “hello”;
if (greeting.equals(“hello”))
{
  System.out.println(“they are equal!”);
}
else
{
  System.out.println(“they aren’t equal!”);
}
```

```
String greeting = “hello”;
if (greeting == “hello”))
{
      // probably true
}
```

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types
- Declaring Variables
- Operators
- String
- Basic IO (console)

Writing Output

- for simple stand-alone java program,

```
System.out.println(data)
```

System.out (standard output) :
a static ***PrintStream*** object declared in class System (java.lang.System)

println method
Print an object (i.e. data) to the standard output stream

Reading Input

- System.in has minimal set of features–it can only read one byte at a time
- In Java 5.0, Scanner class was added to read keyboard input in a convenient manner

```
import java.util.Scanner;
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

Note:
nextDouble reads a double
nextLine reads a line (until user hits Enter)
nextWord reads a word (until any white space)

Java Basics

- (simple) Program Structure
- Comments
- Primitive Data Types
- Declaring Variables
- Operators
- String
- Basic IO (console)
- Control Structures

Control Structures

- Java supports both conditional statements and loops to determine the control flow of a program

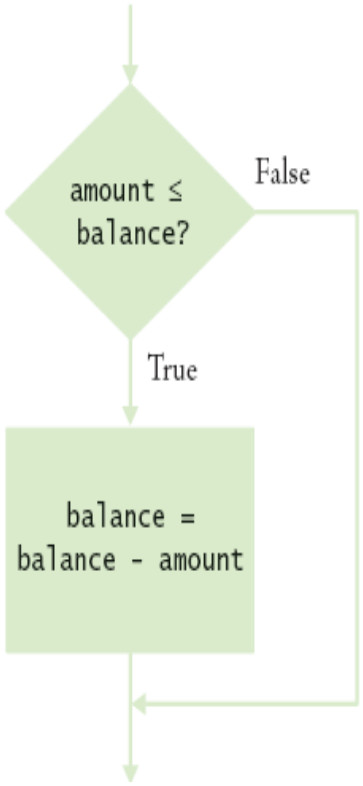


Figure 1
Flowchart for an `if` Statement

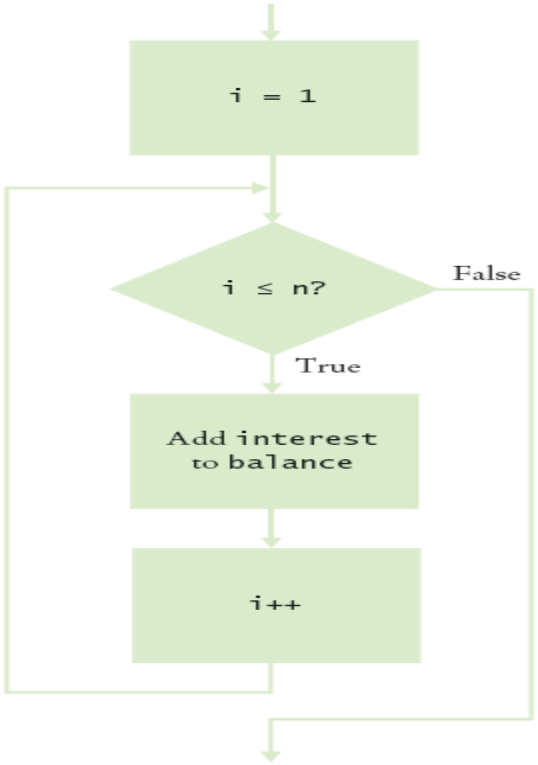
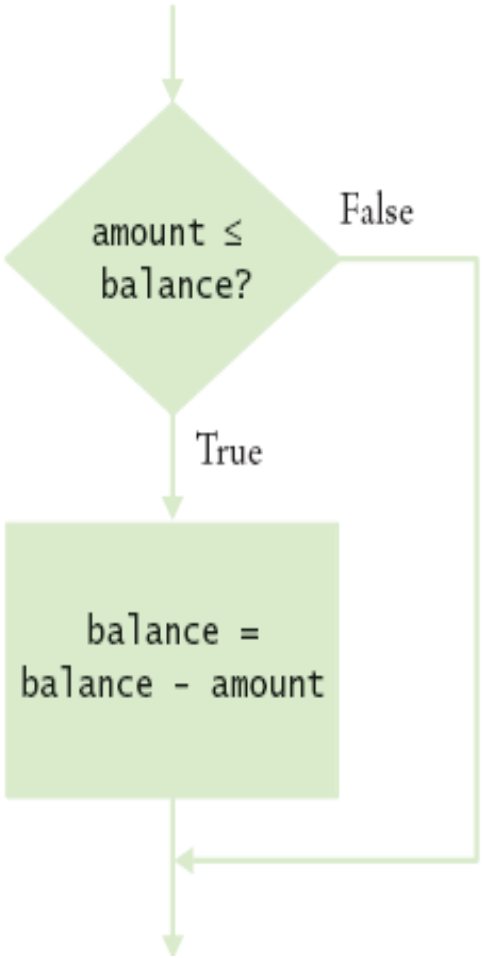


Figure 4 Flowchart of a for Loop

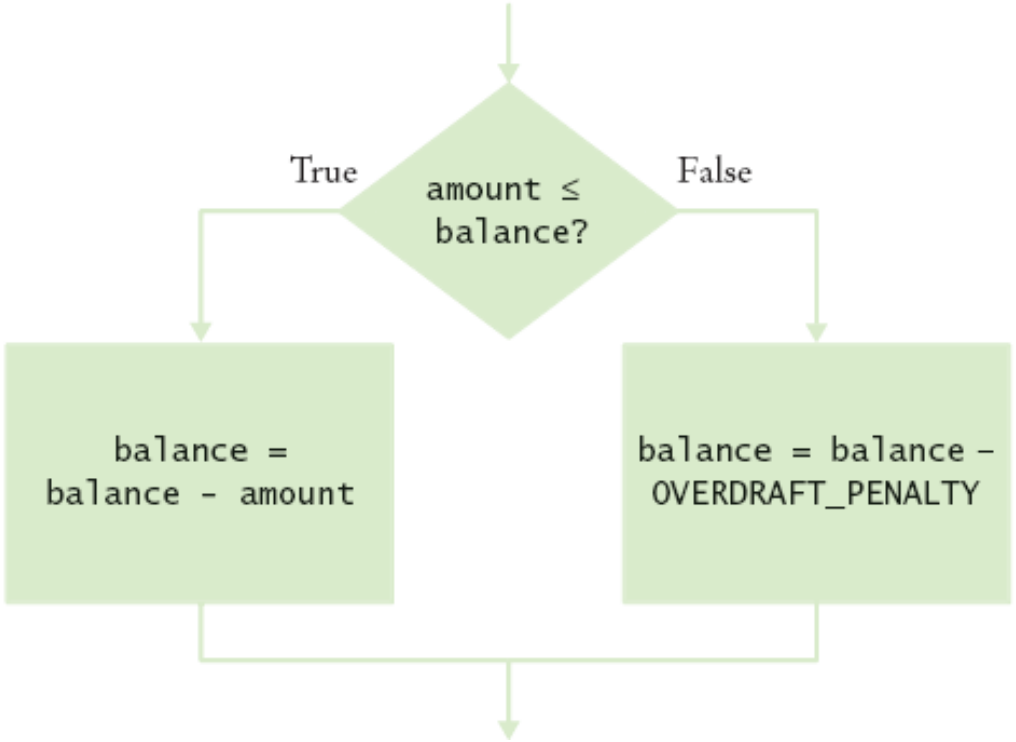
Decisions

- if statement



Decisions

- if/else statement



if statement

Syntax 2.4: if statement

```
if (condition)
{
   statement
}
```

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

Example:

```
if (amount <= balance) balance = balance - amount;
if (amount <= balance)
         balance = balance - amount;
else
         balance = balance - OVERDRAFT_PENALTY;
```

Purpose:

To execute a statement when a condition is true or false

Exercise: implement this loop in Java

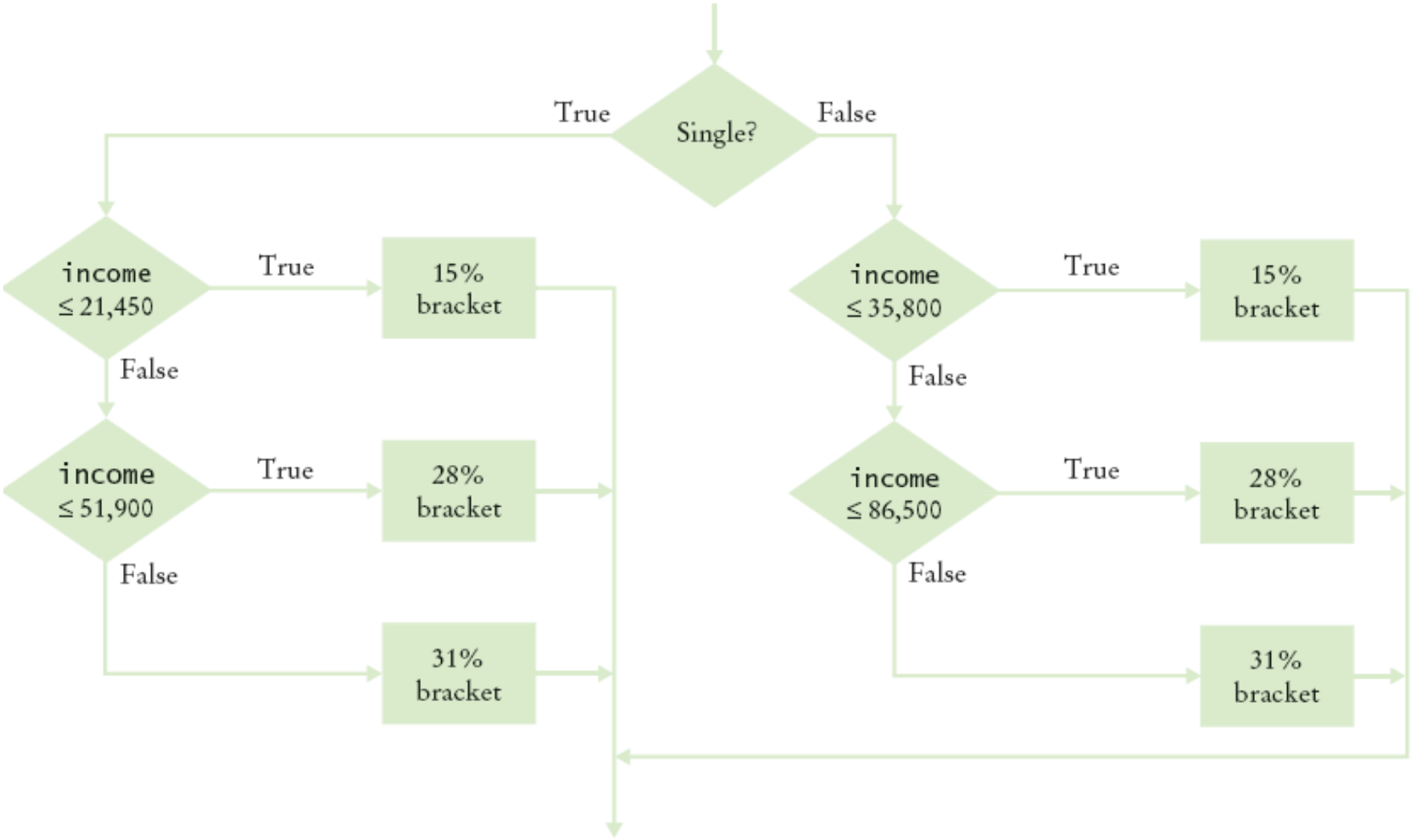


Figure 5 Income Tax Computation Using 1992 Schedule

while loop

- Executes a block of code repeatedly
- A condition controls how often the loop is executed

while (*condition*)
statement;

- Most commonly, the statement is a block statement (set of statements delimited by { })

while loop

Calculating the Growth of an Investment
Invest $10,000, 5% interest, compounded annually

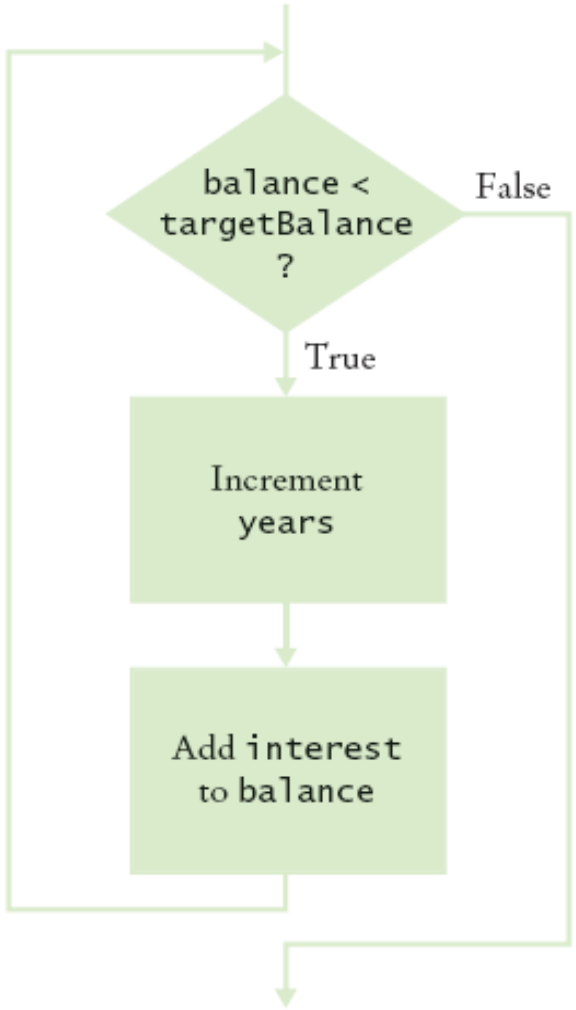
Year	Balance
0	$10,000
1	$10,500
2	$11,025
3	$11,576.25
4	$12,155.06
5	$12,762.82

When has the bank account reached a target balance of $500,000 ?

while loop

Calculating the Growth of an Investment
Invest $10,000, 5% interest, compounded annually

When has the bank account reached a target balance of $500,000 ?



while statement

Syntax 2.5: while statement

while (*condition*)
  statement

Example:

```
while (balance < targetBalance)
{
        year++;
        double interest = balance * rate / 100;
        balance = balance + interest;
}
```

Purpose:

To repeatedly execute a statement as long as a condition is true

for loop

for (*initialization*; *condition*; *update*)

statement

Example:

```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

equivalent to

initialization;

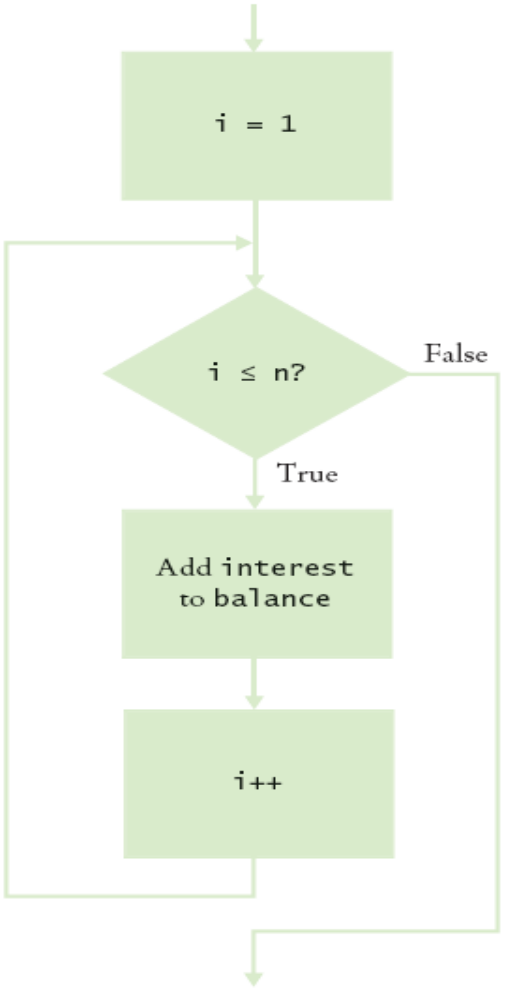
while (*condition*) { *statement*; *update*; }

Examples:

for (years = n; years > 0; years--) . . .

for (x = -10; x <= 10; x = x + 0.5) . . .

for loop




```
for (int i = 1; i <= n; i++)
{
        double interest = balance * rate / 100;
        balance = balance + interest;
}
```

for statement

Syntax 2.6: for statement

for (*initialization*; *condition*; *update*)
  statement

Example:

```
 for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

Purpose:

To execute an initialization, then keep executing a statement and updating an expression while a condition is true

Summary

- fundamentals of Java language
  - program structure, comment
  - primitive data types
  - variables
  - control flows

- Java API's classes: Math , String
- Console I/O
  - Input (Scanner class)
  - Output (System.out class)

Additional Readings

- The Java Tutorials : Getting Started,

  http://docs.oracle.com/javase/tutorial/getStarted/TOC.html
- The Java Language Environment,

  http://www.oracle.com/technetwork/java/langenv-140151.html