Course Code	Course Title
BTCS14F6550	Programming with Java

UNIT - 1

Primitive Data Types and Arithmetic: Data, Data Storage, Identifiers, Syntax, Variables and Constants, the Format of a Simple Program, Arithmetic, Operator Precedence, Casting,

Objects: Introduction to Objects, The String Class, The Anatomy of a Simple Program Revisited, The AVI Package, The Window Class, Input to a Dialog Box, Converting Strings to Numbers, Command Line Arguments, Errors

UNIT - 2

Object-Oriented Programming: Abstract Data Type, Constructors, Instance Methods, Class Methods, Scope and Lifetime of Identifiers, Software Development, Object-Oriented Program Design, the AVI Package Revisited

Selection: More AVI Classes, If..else Statement, Nested If Statement, Conditional Expressions, Else if 1Statements, Boolean Data Type, Switch, Wrapper Classes, Yet another AVI Class!, The This Object.

UNIT - 3

Repetition and One-Dimensional Arrays: Loop Structure, While Loop, Do..while Loop, Increment/Decrement Operators, For Loop, Which Loop?, Arrays Revisited, Declaring and Initializing One-Dimensional Arrays, Using Arrays, Our Last AVI Class: CheckBoxes, Formatting Numbers for Output

Advanced Concepts with Classes: Inheritance, An Example of Inheritance, Overriding Superclass Methods, Polymorphism, Instanceof Operator, Shadowed Variables, Inner Classes, Abstract Methods and Classes, Interfaces, Constructors Revisited, Instance Methods Revisited, Object Properties, Comparing Objects, Copying Objects, Passing Objects as Parameters, Garbage Collection and Object Finalization

UNIT - 4

Exceptions and Streams: Introduction, Exception Classes, Catching an Exception, Catching Multiple Exceptions, Creating Your Own Exception Class, Throwing an Exception, Finally Blocks, Using Exception Handling, Stream Input and Output, The StreamTokenizer Class, Text File Processing, The FileDialog,

Applets and Threads: Introduction, Applets, Input to Applets, Playing Sounds, Displaying Images, Loading Images, Arrays Revisited, Image Maps, Threads, Animation, Restrictions, Sound and Images with Applications

Recommended Learning Resources:

1. Barry J. Holmes and Daniel T. Joyce, Object-Oriented Programming With Java; Second Edition;Jones And Bartlett Publishers,2000
2. Dale Skrien;Object-Oriented Design Using Java; McGraw-Hill Higher Education; 2009
3. Danny Poo; Object-Oriented Programming and Java; Second Edition; Springer; 2008.

References:

1. Cay Horstmann; Big Java; 2nd Edition ; John Wiley and Sons
2. Herbert Schildt; The Complete Reference Java J2SE; 5th Edition; TMH Publishing Company Ltd, New Delhi
3. H.M.Dietel and P.J.Dietel; Java How to Program; Sixth Edition; Pearson Education/PHI
4. Cay.S.Horstmann and Gary Cornell; Core Java 2, Vol 1, Fundamentals; Seventh Edition; Pearson Education/PHI
5. Cay.S.Horstmann and Gary Cornell; Core Java 2, Vol 2, Advanced Features; Seventh Edition; Pearson Education/PHI
6. Beginning in Java 2 by Iver Horton, Wrox Publications.

Java History

- Computer language innovation and development occurs for two fundamental reasons:

  1)to adapt to changing environments and uses

  2)to implement improvements in the art of programming

- Many Java features are inherited from the earlier languages:

  B → C → C++ → Java

Before Java: C

- Designed by Dennis Ritchie in 1970s.
- Before C: BASIC, COBOL, FORTRAN, PASCAL
- C- structured, efficient, high-level language that could replace assembly code when creating systems programs.
- Designed, implemented and tested by programmers.

Before Java: C++

- **Designed by Bjarne Stroustrup in 1979.**
- **Response to the increased complexity of programs and respective improvements in the programming paradigms and methods:**
    1) **assembler languages**
    2) **high-level languages**
    3) **structured programming**
    4) **object-oriented programming (OOP)**
- **OOP – methodology that helps organize complex programs through the use of inheritance, encapsulation and polymorphism.**
- **C++ extends C by adding object-oriented features.**

Java: History

- **In 1990, Sun Microsystems started a project called Green.**
- **Objective: to develop software for consumer electronics.**
- **Project was assigned to James Gosling, a veteran of classic network software design. Others included Patrick Naughton, ChrisWarth, Ed Frank, and Mike Sheridan.**
- **The team started writing programs in C++ for embedding into**
  - **– toasters**
  - **– washing machines**
  - **– VCR's**
- **Aim was to make these appliances more "intelligent".**

Java: History (contd.)

- **C++ is powerful, but also dangerous. The power and popularity of C derived from the extensive use of pointers. However, any incorrect use of pointers can cause memory leaks, leading the program to crash.**
- **In a complex program, such memory leaks are often hard to detect.**
- **However, users do not expect toasters to crash, or washing machines to crash.**
- **A design for consumer electronics has to be *robust*.**
- **Replacing pointers by references, and automating memory management was the proposed solution.**

Java: History (contd.)

- **Hence, the team built a new programming language called Oak, which avoided potentially dangerous constructs in C++, such as pointers, pointer arithmetic, operator overloading etc.**
- **Introduced automatic memory management, freeing the programmer to concentrate on other things.**
- **Architecture neutrality (Platform independence)**
- **Many different CPU's are used as controllers.**
- **So, the software and programming language had to be** *architecture neutral***.**

Java: History (contd)

- **It was soon realized that these design goals of consumer electronics perfectly suited an ideal programming language for the Internet and WWW, which should be:**
  - **object-oriented (& support GUI)**
  - **– robust**
  - **– architecture neutral**
- **Internet programming presented a BIG business opportunity. Much bigger than programming for consumer electronics.**
- **Java was "re-targeted" for the Internet**
- **The team was expanded to include Bill Joy (developer of Unix), Arthur van Hoff, Jonathan Payne, Frank Yellin, Tim Lindholm etc.**
- **In 1994, an early web browser called WebRunner was written in Oak. WebRunner was later renamed HotJava.**
- **In 1995, Oak was renamed Java.**
- **A common story is that the name Java relates to the place from where the development team got its coffee. The name Java survived the trade mark search.**

Java History

- Designed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.
- The original motivation is not Internet: platform-independent software embedded in consumer electronics devices.
- With Internet, the urgent need appeared to break the fortified positions of Intel, Macintosh and Unix programmer communities.
- Java as an “Internet version of C++”? No.
- Java was not designed to replace C++, but to solve a different set of problems.

The Java Programming Language

- **Robust** and **Versatile** enabling developers to:
  - Write software on one platform and run it on another.
  - Create programs to run within a web browser.
  - Develop server-side applications for online forums, stores, polls, processing HTML forms, and more.
  - Write applications for cell phones, two-way pagers, and other consumer devices.

The Java Buzzwords

- The key considerations were summed up by the Java team in the following list of buzzwords:
  - ❖ Simple
  - ❖ Secure
  - ❖ Portable
  - ❖ Object-oriented
  - ❖ Robust
  - ❖ Multithreaded
  - ❖ Architecture-neutral
  - ❖ Interpreted
  - ❖ High performance
  - ❖ Distributed
  - ❖ Dynamic

Java Technology

- Java technology is both a *programming language* and a platform
- The Java programming language is a high-level language.

Simple

- Primary characteristics of the Java programming language include a ***simple*** language that can be programmed without extensive programmer training.

- The fundamental concepts of Java technology are grasped quickly; programmers can be productive from the very beginning

Object Oriented

- *Java technology* provides a clean and efficient object-based development platform. *Java programming language* is designed to be *object oriented* from the ground up.

Portable & Distributed

- Java technology is designed to support applications that will be deployed into heterogeneous network environments.
  - Applications must be capable of executing on a variety of hardware architectures
  - The Java Compiler generates ***bytecodes*** (an *architecture neutral)* to supply the diversity of operating environments
    - ✓ Intermediate format designed to transport code efficiently to multiple hardware and software platforms.
    - ✓ The same Java programming language byte codes will run on any platform.

Architecture Neutrality

- Just one part of a truly ***portable*** system
  - ❖ The programs are the same on every platform
    - ✓ There are no data type incompatibilities across hardware and software architectures.

- The architecture-neutral and portable language platform of Java technology is known as the ***Java virtual machine*****.**
  - ❖ Java programming language compilers generate code to specifiy an abstract machine. Then;
    - ✓ Specific implementations of the Java virtual machine provide the concrete realization of the virtual machine for specific hardware and software platforms.
  - ❖ The Java virtual machine is based primarily on the POSIX interface specification (an industry)
    - ✓ Standard definition of a portable system interface.

High Performance

- ***Performance*** is always a consideration.
- The interpreter can run at full speed without needing to check the run-time environment.
  - The Java platform achieves superior performance by adopting a scheme
- The ***automatic garbage collector*** runs as a low-priority background thread
  - Ensuring a high probability that memory is available when required,
  - Leading to better performance.
- Applications requiring large amounts of compute power can be designed such that
  - Compute-intensive sections can be rewritten in native machine code as required and interfaced with the Java platform.
- In general, interactive applications respond quickly even though they're interpreted.

Interpreted & Threaded

- The *Java interpreter* can execute Java bytecodes **directly** on any machine to which the interpreter and run-time system have been ported.
- Modern network-based applications, such as the HotJava Browser for the WWW, can do several things at the same time
  - A user working with HotJava Browser can run several animations concurrently while downloading an image and scrolling the page.
- Java technology's ***multithreading*** capability provides to build applications with many concurrent threads of activity.
  - Thus, multithreading results in a high degree of interactivity for the end user.
- The Java platform supports **multithreading** at the language level with the addition of sophisticated synchronization primitives:
  - The language library provides the Thread class,
  - The run-time system provides monitor and condition lock primitives.

Dynamic

- Java Language and run-time system are ***dynamic*** in their linking stages; but
  - The Java Compiler is strict in its compile-time static checking
- Classes are linked only as needed.
  - New code modules can be linked in on demand from a variety of sources, even from sources across a network.

The Requirements to Write Any Java Program

We need two items to write a Java program for the Java 2 Platform, Standard Edition

- The Java2 Platform, Standard Edition http://java.sun.com/j2se/1.4.2/download.html

- A text editor. We can use the simple editor NotePad included with the Windows platforms.

Creating the First Java Application

- **Create a source file.**

  A source file contains text, written in the Java programming language. We can use any text editor to create and edit source files.

- **Compile the source file into a bytecode file**

  The *compiler*, javac, takes the source file and translates its text into instructions that the *Java Virtual Machine* (Java VM) can understand. *In other words*:

  - The compiler converts these instructions into a bytecode file.

- **Run the program contained in the bytecode file.**

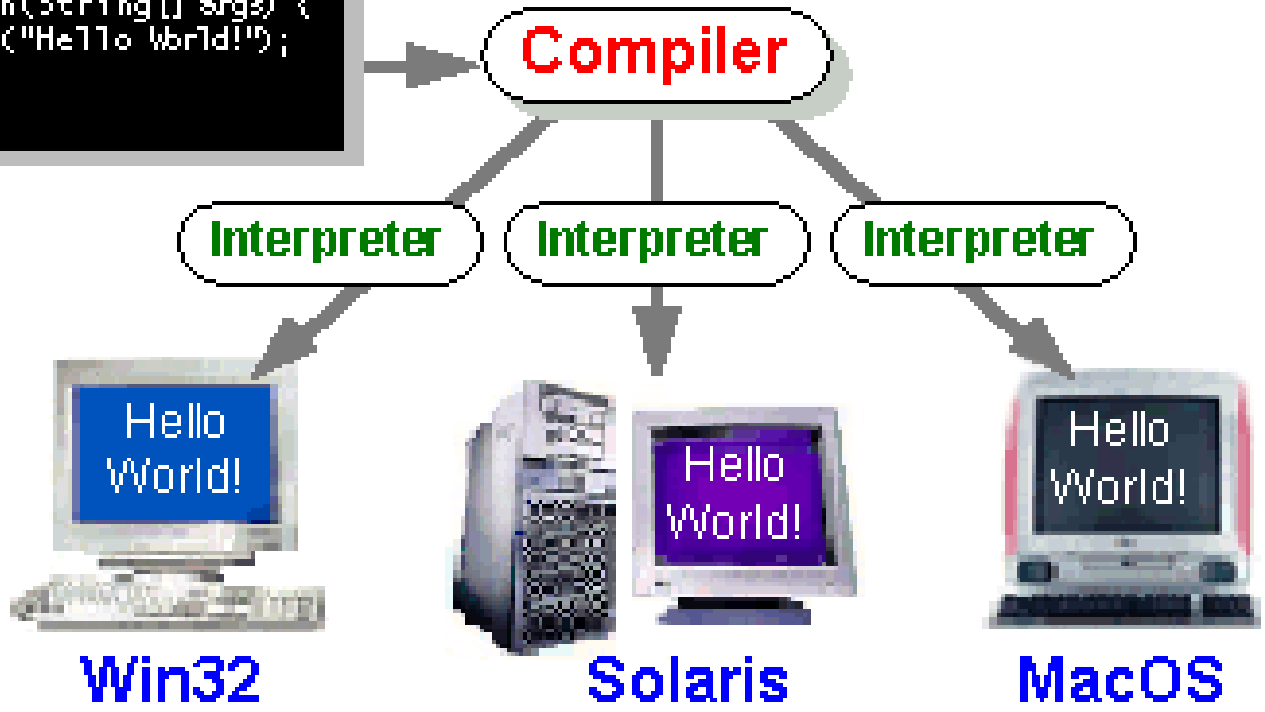
  The Java interpreter installed on the computer implements the Java VM.

  - This interpreter takes the bytecode file and carries out the instructions by translating them into instructions that the computer can understand

Java Program

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

HelloWorldApp.java



Write Once and Run Everywhere

This important and famous property of the Java programming language means that:

- When we compile our program, we don't generate instructions for one specific platform.
  - we generate Java bytecodes, which are instructions for the Java Virtual Machine (Java VM).

- Whether our platform is Windows, UNIX, MacOS, or an Internet browser, it has the Java VM
  - it can understand those bytecodes.

Two Paradigms for Computer Programs

There are two paradigms for a program construction:

- Process Oriented Model
  - This model can be thought of as ***code acting on data***
    - ✓ Procedural languages, such as C, characterize a series of linear steps (that is, code)

- Object- Oriented Programming
  - Aims to manage increasing complexity.
  - Organizes a program around its data (that is object) and a set of well-defined interfaces to that data.
    - ✓ This program can be characterized as ***data controlling access to code.***

Some of the Object-Oriented Paradigm are:

1. Emphasis is on data rather than procedure.
2. Programs are divided into objects.
3. Data Structures are designed such that they Characterize the objects.
4. Methods that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can not be accessed by external functions.
6. Objects may communicate with each other through methods.

Basic Building Blocks of OOPS

The following are the basic oops concepts:
They are as follows:

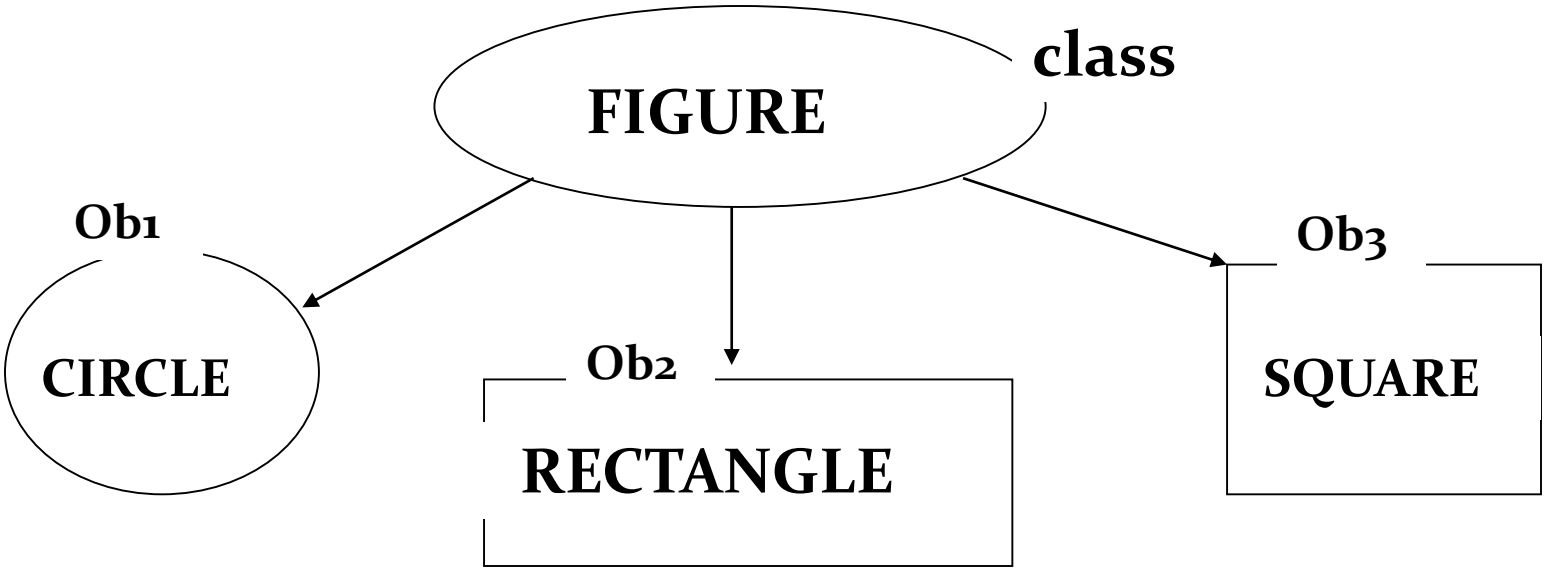
1. Objects.
2. Classes.
3. Data Abstraction.
4. Data Encapsulation.
5. Inheritance.
6. Polymorphism.
7. Dynamic Binding.
8. Message Passing.

INSTANCE

- Instance is an Object of a class which is an entity with its own attribute values and methods.

CLASSES

- Class is blue print or an idea of an Object
- From One class any number of Instances can be created
- It is an encapsulation of attributes and methods



Abstraction

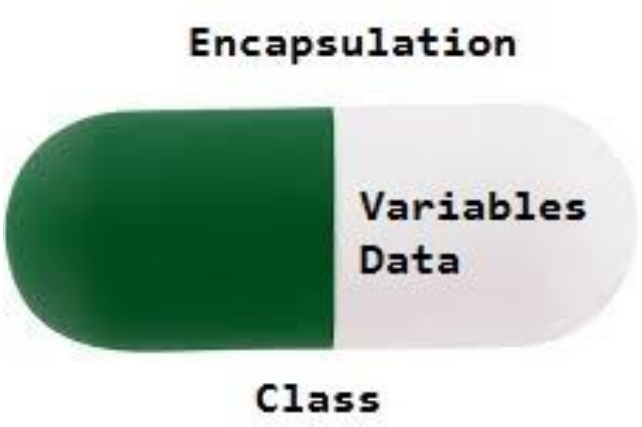
The essential element of OOP is ***abstraction***

- It is the process of focusing on those *features of something* that are essentials for the task at hand and ignoring those that are not.
- The powerful way to manage abstraction is through the use of hierarchical classifications
  - This allows us to layer semantic of complex systems, breaking them into more manageable pieces.
    - ✓From the outside, the car is a single object. Once inside, we see that car consists of several subsystems: steering, clutch pedal, brakes, sound system, seat belts, heading,, and so on.
    - ✓Each subsystem is made of more specialized units.
    - ✓The sound system consists of a radio, CD player, a tape player

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.



Encapsulation (con’t)

Advantages:

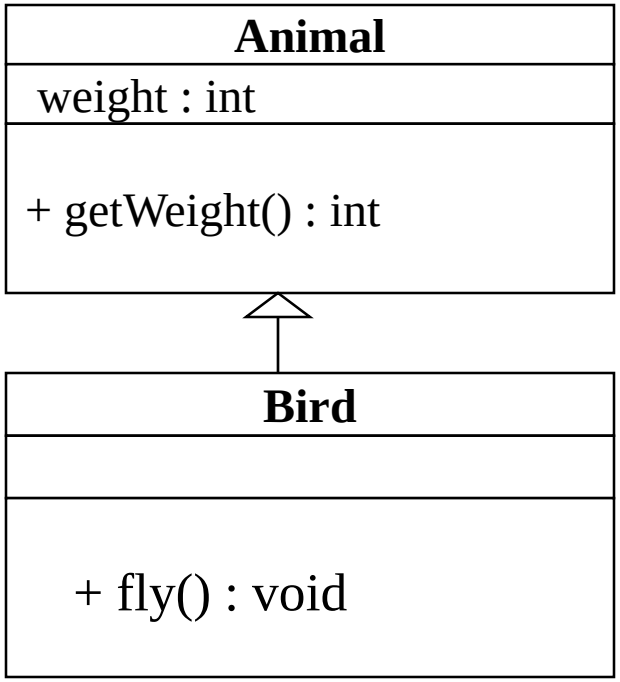
- Data Hiding
- Increased Flexibility
- Reusability
- Testing code is easy

Inheritance

- Methods allows to reuse a sequence of statements
- *Inheritance* allows to reuse classes by deriving a new class from an existing one
- The existing class is called the *parent class*, or *superclass*, or *base class*
- The derived class is called the *child class* or *subclass*.
- The child class inherits characteristics of the parent class(i.e the child class *inherits* the methods and data defined for the parent class

Inheritance

- Inheritance relationships are often shown graphically in a *class diagram*, with the arrow pointing to the parent class



Methods and Properties

- Some of the information in the **object** may actually be **directly accessible**
- Other information may require us to use a ***method*** to access it .

Because:

- The information stored internally may be unnecessary for the user and
- Only certain things can be written into the information space, and the object needs to check whether the program is outside the limits.

Methods and Properties (con't)

- The directly accessible bits of information in the object are its ***properties*.**

- The difference between data accessed via properties and data accessed via methods is that:

  - *with properties*

    you see exactly what you're doing to the object;

  - with methods,

    unless you created the object yourself, you just see the effects of what you're doing.

Primitive Data Types and Arithmetic

By the end of the chapter you will have an understanding of the following topics.

- Recognizing data and classifying it by type
- The identification of variables and constants and their representation in a program
- The construction of arithmetic expressions for the purpose of making calcula-tions
- Writing simple programs

1.1 Data

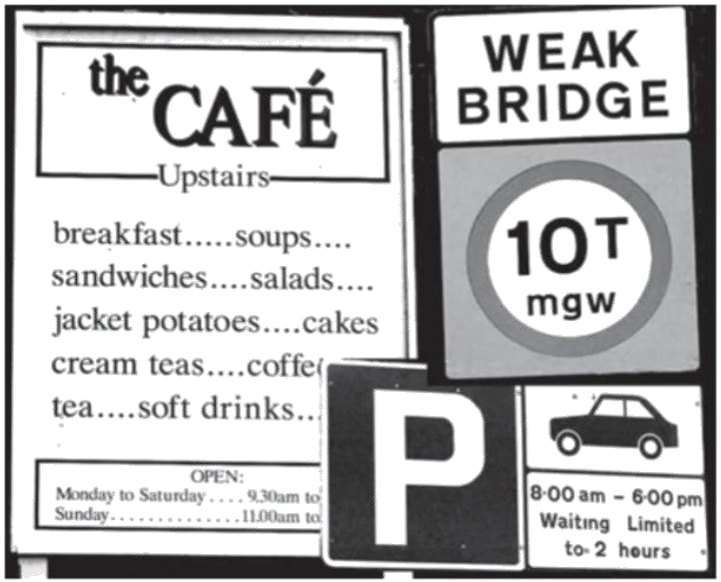


Figure 1.1 : An assortment of signs

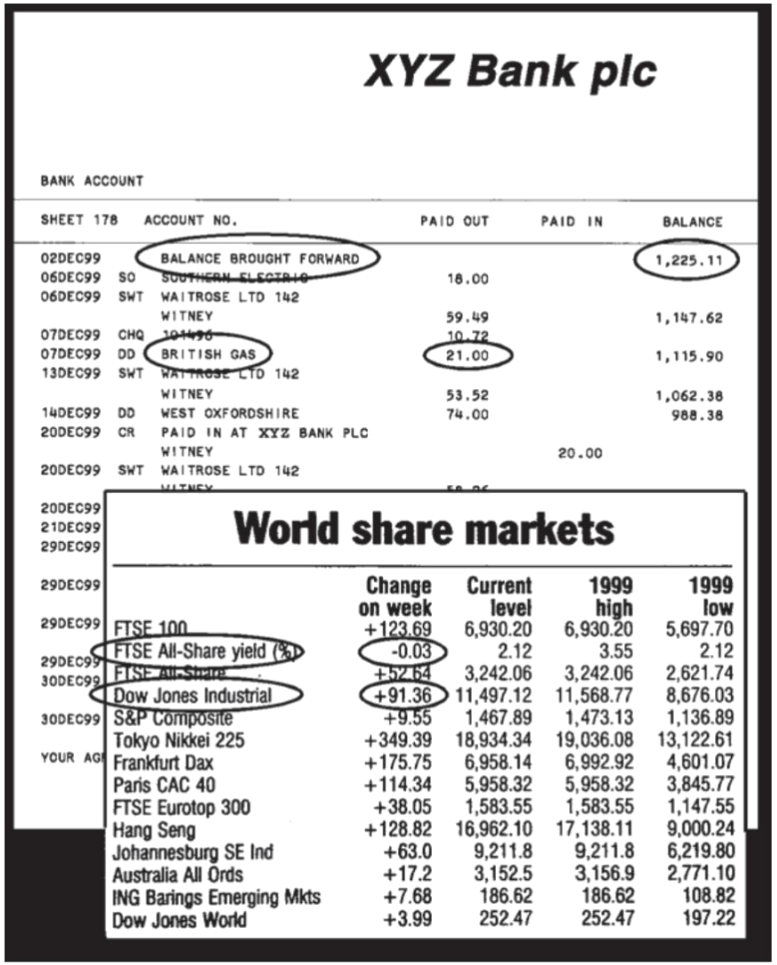
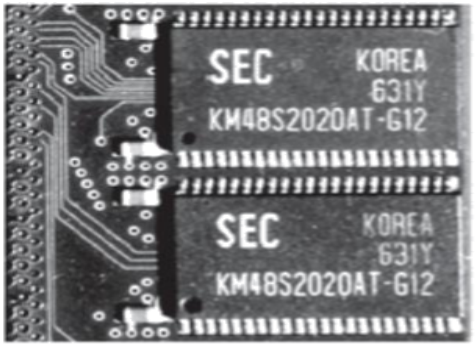


Figure 1.2 : Information from a newspaper and bank statement

1.1 Data

- **Data** are simply facts or figures — bits of **information**, but not **information** itself.
- When **data** are processed, interpreted, organized, structured or presented so as to make them meaningful or useful, they are called i**nformation**. **Information** provides context for **data**.

1.2 Data Storage



The smallest memory component can physically represent a binary digit (**bit**). These components are grouped together in units of eight to form a **byte** of memory. Each memory chip shown here has a storage capacity of over four million bytes.

Each byte of memory has a unique numeric address to enable data to be written to or read from specific locations in memory.

A binary digit is stored as a representation of some physical property of the circuitry being used, for example as levels of electrical charge.

The diagram shows that at any memory address the eight bits are represented in one of two states as either a peak or a trough, where a peak represents the binary digit 1 and a trough the binary digit 0. For example the contents of address 20003 can be interpreted as the binary value 11010101 .

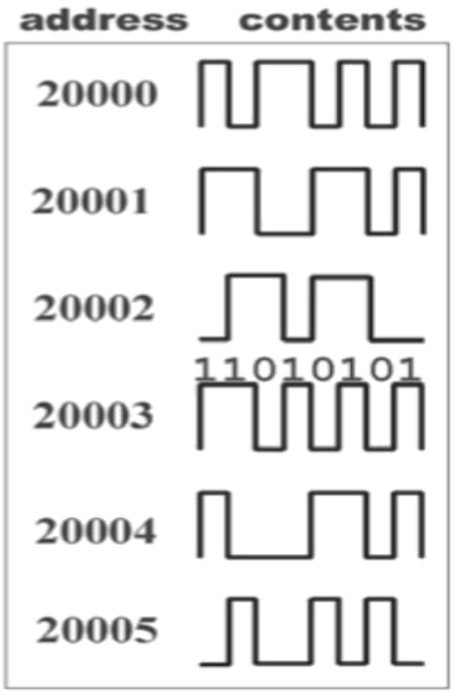


Figure 1.3 : Computer memory

- **JDK**

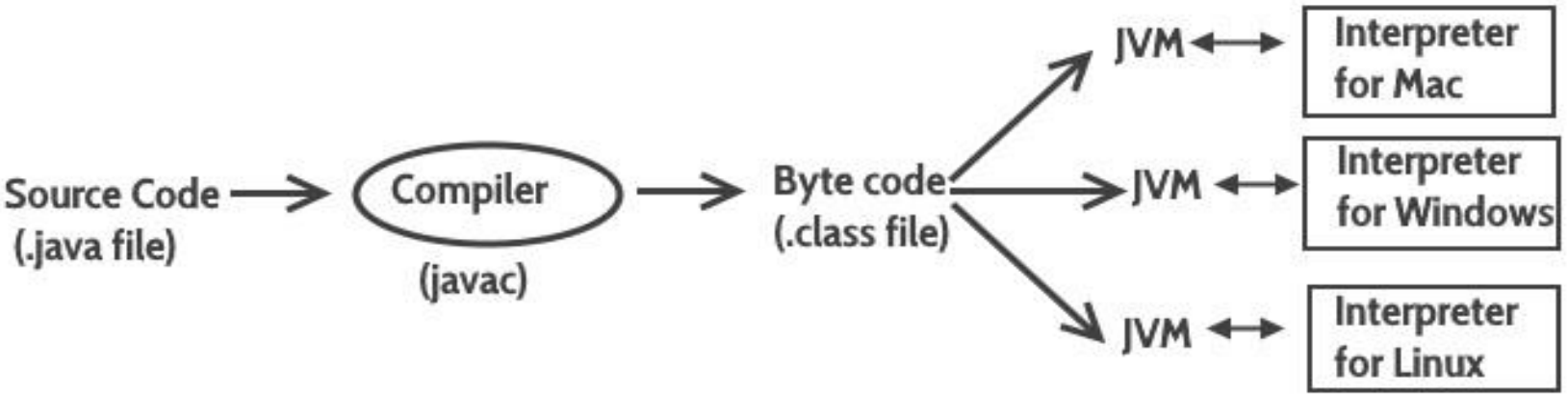
  JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

- **Java Runtime Environment(JRE)**

  JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on your system, you can run a java program however you won't be able to compile it. JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

- Java is a high level programming language. A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The **javac compiler** does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).

- **Java Virtual Machine (JVM)** is a virtual machine that resides in the real machine (your computer) and the **machine language for JVM is byte code**. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. **JVM is the one that makes java platform independent**.



Binary Digit

- A *binary digit* or *bit* has one of two possible values and is the smallest unit of memory available on a computer.

  Eg., on or off, open or closed, high charge or low charge
- To represent more than two values we need to use more bits.

  For example, we could classify the temperature into four categories by using two bits as follows: 00 = cold, 01 = pleasant, 10 = warm, and 11 = hot.
- Every time we add a bit, we double the number of values we can represent.

  With three bits we can represent $2^3 = 8$ values (000, 001, 010, 011, 100, 101, 110, 111), with four bits we can represent $2^4 = 16$ values, and so on.
- A collection of eight bits is called a *byte* and can represent $2^8 = 256$ values. Bytes are often used as the unit of addressing in computers, and therefore we describe the size of computer memories in terms of bytes.

Number Systems

with computers it is more efficient to use the *binary number system* (base 2) for calculations inside the machine.

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Figure 1.4 Representation of 0 to 15 in binary, decimal, and hexadecimal

Data Types

- Java defines eight simple types:
    1)byte – 8-bit integer type
    2)short – 16-bit integer type
    3)int – 32-bit integer type
    4)long – 64-bit integer type
    5)float – 32-bit floating-point type
    6)double – 64-bit floating-point type
    7)char – symbols in a character set
    8)boolean – logical values true and false

- byte: 8-bit integer type.

  Range: -128 to 127.

  Example: byte b = -15;

  Usage: particularly when working with data streams.

- short: 16-bit integer type.

  Range: -32768 to 32767.

  Example: short c = 1000;

  Usage: probably the least used simple type.

- **int:** 32-bit integer type.

  Range: -2147483648 to 2147483647.

  Example: int b = -50000;

  Usage:

  1) Most common integer type.

  2) Typically used to control loops and to index arrays.

  3) Expressions involving the byte, short and int values are promoted to int before calculation.

- **long:** 64-bit integer type.

  Range: -9223372036854775808 to 9223372036854775807.

  Example: long l = 10000000000000000;

  Usage: 1) useful when int type is not large enough to hold the desired value

- **float:** 32-bit floating-point number.

  Range: 1.4e-045 to 3.4e+038.

  Example: float f = 1.5;

  Usage:

  1) fractional part is needed

  2) large degree of precision is not required

- **double:** 64-bit floating-point number.

  Range: 4.9e-324 to 1.8e+308.

  Example: double pi = 3.1416;

  Usage:

  1) accuracy over many iterative calculations

  2) manipulation of large-valued numbers

char: 8-bit data type used to store characters.

Range: 0 to 65536.

Example: char c = ‘a’;

Usage:

1) Represents both ASCII and Unicode character sets; Unicode defines a

character set with characters found in (almost) all human languages.

2) Not the same as in C/C++ where char is 8-bit and represents ASCII only.

- **boolean:** Two-valued type of logical values.

  Range: values true and false.

  Example: boolean b = (1<2);

  Usage:

  1) returned by relational operators, such as 1<2

  2) required by branching expressions such as if or for

1.3 Identifiers

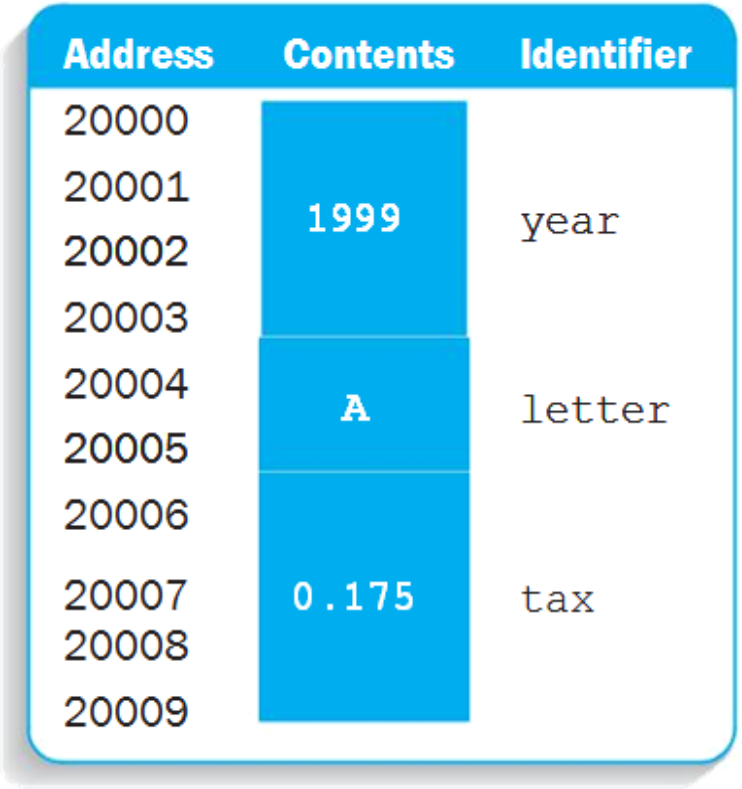


Figure 1.6 Use of identifiers to represent data

1.3 Identifiers Rules

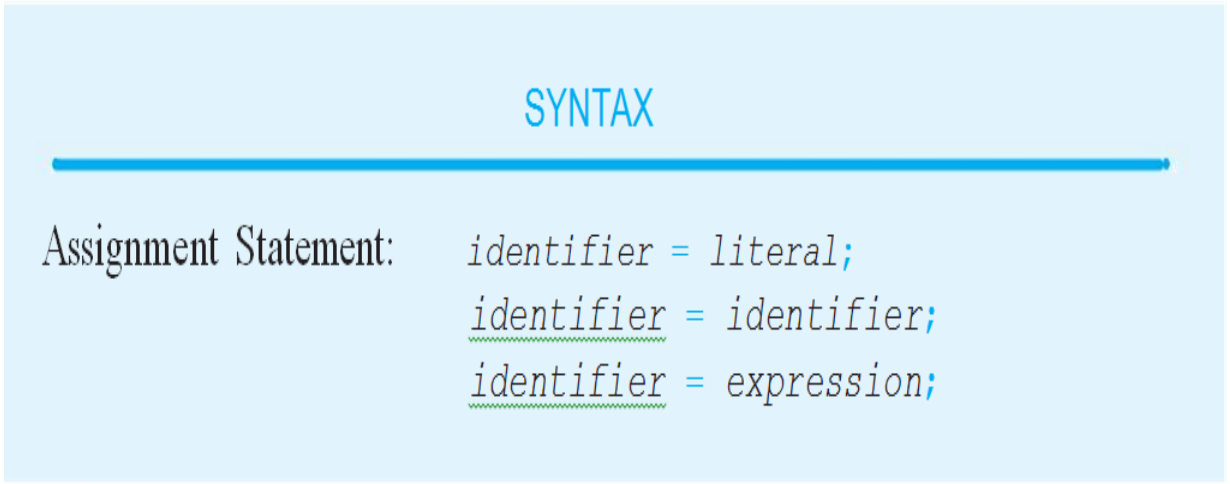
- An identifier may contain combinations of the letters of the alphabet (both uppercase A-Z and lowercase a-z), an underscore character _, a dollar sign $, and decimal digits 0-9.
- The identifier may start with any of these characters with the exception of a decimal digit.
- Java is a *case-sensitive* language, meaning that uppercase letters and lowercase letters of the alphabet are treated as different letters. Identifiers can normally be of any practicable length. An identifier must not be the same as those Java key-words.

JAVA KEYWORDS

abstract	default	goto	operator	synchronized
boolean	do	if	outer	this
break	double	implements	package	throw
byte	else	import	private	throws
byvalue	extends	inner	protected	transient
case	false	instanceof	public	true
cast	final	int	rest	try
catch	finally	interface	return	var
char	float	long	short	void
class	for	native	static	volatile
const	future	new	super	while
continue	generic	null	switch	

Figure 1.7 Keywords

1.4 Syntax to use identifier



tax = 135.86;	where 135.86 is a numeric literal
tax = incomeTax;	where incomeTax is another identifier
tax = 0.175*cost;	where 0.175*cost is an expression

By using the rules of assignment, it is easy to understand that the following statements are illegal.

135.86 = tax;	this implies the syntax *literal =identifier* (wrong)
tax = incomeTax	statement delimiter ; missing (wrong)
0.175*cost = tax;	this implies *expression = identifier* (wrong)

1.5 Variables and Constants

- declaration – how to assign a type to a variable
- initialization – how to give an initial value to a variable
- scope – how the variable is visible to other parts of the program
- lifetime – how the variable is created, used and destroyed
- type conversion – how Java handles automatic type conversion
- type casting – how the type of a variable can be narrowed down
- type promotion – how the type of a variable can be expanded

Variables

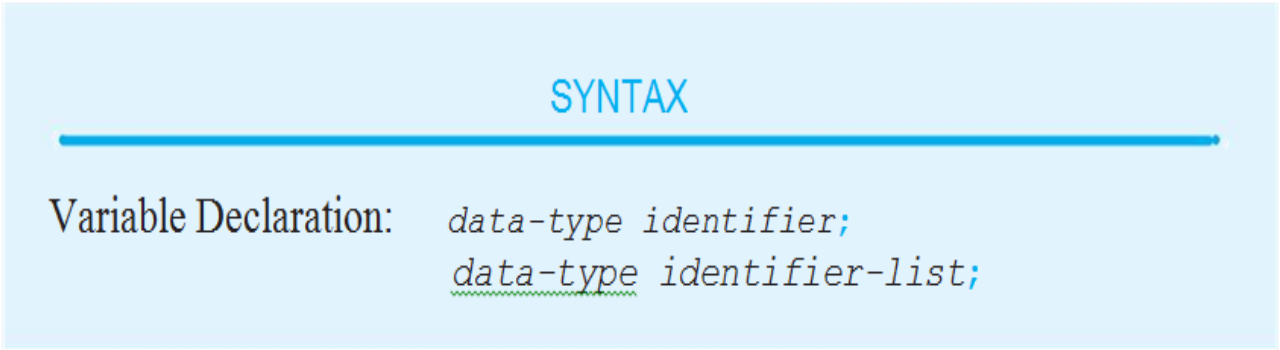
- Java uses variables to store data.
- To allocate memory space for a variable JVM requires:

  1) to specify the data type of the variable

  2) to associate an identifier with the variable

  3) optionally, the variable may be assigned an initial value
- All done as part of variable declaration.

Syntax of Basic Variable Declaration



For example,

```
Int weightLimit;
Int parkingTime;
float balance;
float costOfGas;
float valueOfShares;
```

When variables are of the same type, you may declare the type followed by a list of identifiers separated by commas, for example:

```
Int weightLimit, parkingTime;
float balance, costOfGas, valueOfShares;
```

Syntax of Basic Variable Initialization

SYNTAX

Variable Initialization: *data-type identifier = literal;*

For example,

```
char    parkingSymbol = 'P';
int     weightLimit   = 10;
float   balance       = 1225.11f;
```

Syntax of Constant Declaration

SYNTAX

Constant Declaration: `final data-type identifier = literal;`

Such constants can be declared in Java as follows.

```
final float  SALES_TAX = 0.05f;
final double PI        = 3.14159;
final float  G         = 9.80665f;
```

1.6 The Format of a Simple Program

```
heading giving details of the name and purpose of the program

import list

class name
{
        main method
        {
                declarations of constants
                declarations of variables
                program statements
        }
}
```

Let's create the hello java program:

```
class Simple{

    public static void main(String args[]){

     System.out.println("Hello Java");

    }
}
```

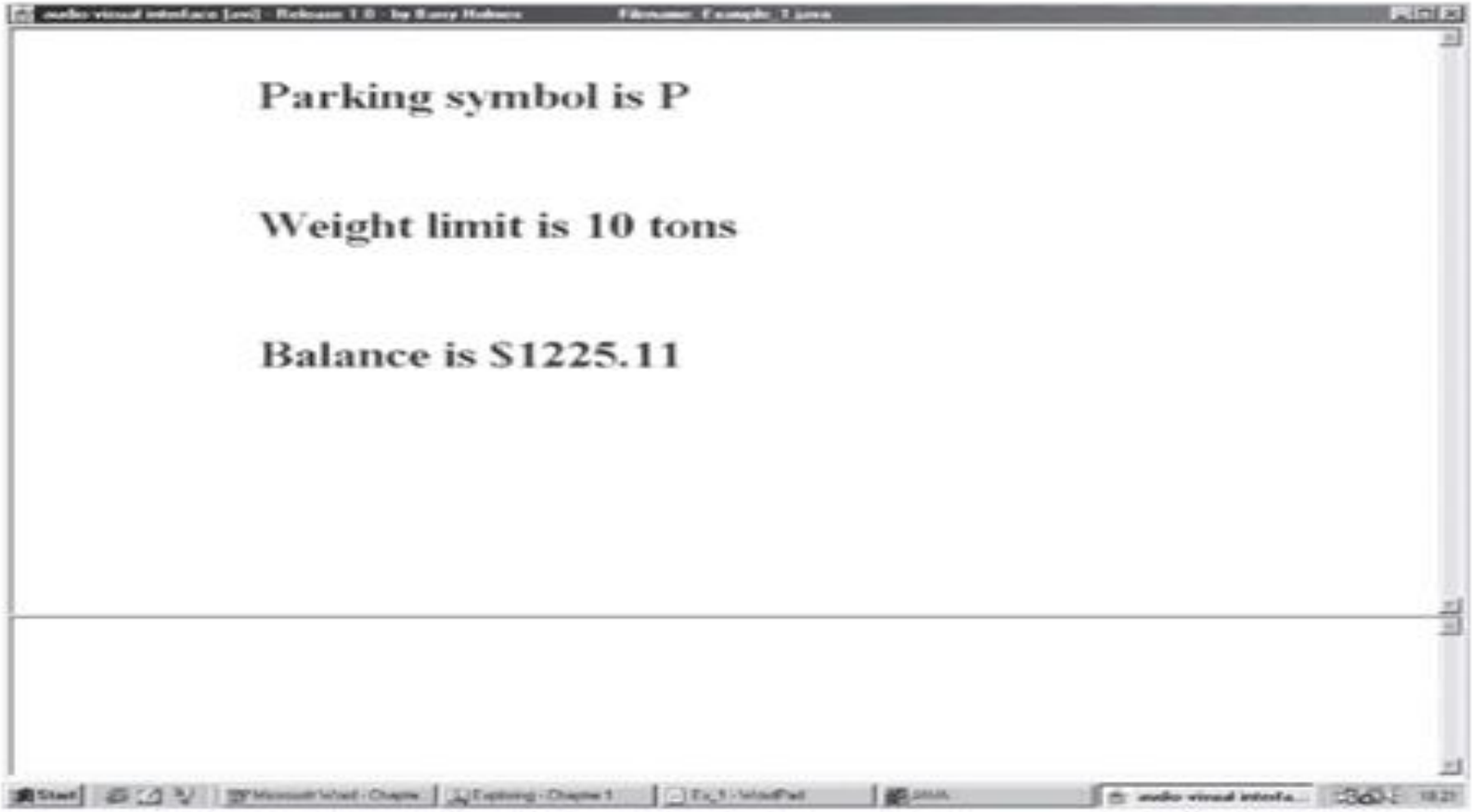
- Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

Program to illustrate the declaration and initialization of variables

```
import avi.*;
class Example_1
{
public static void main(String[] args)
{
char parkingSymbol = 'P';
int weightLimit = 10;
float balance = 1225.11f;
Window screen = new Window("Example_1.java","bold","blue",36);
screen.showWindow();
screen.write("\n\tParking symbol is "+parkingSymbol+"\n\n");
screen.write("\n\tWeight limit is "+weightLimit+"tons\n\n");
screen.write("\n\tBalance is $"+balance+"\n");
}
}
```

The following screen shot shows the program's results.



1.7 Arithmetic

Unary Operators

Unary operators have one operand and are used to represent positive or negative numbers.

Binary Multiplicative Operators

* multiplication

/ division

% remainder

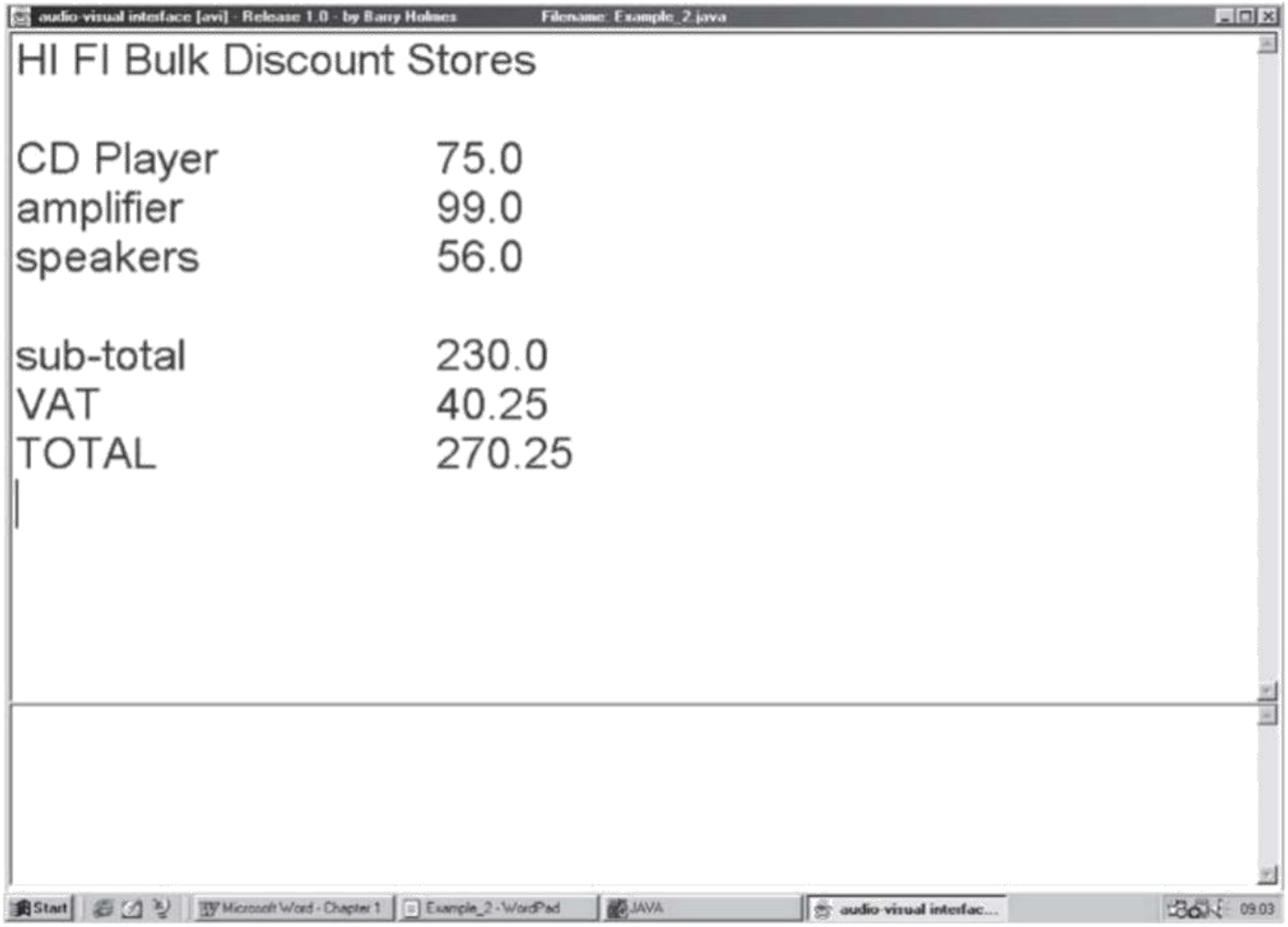
Binary Additive Operators

- addition
- subtraction

```
program to illustrate the use of arithmetic statements
import avi.*;
public class Example_2
{
public static void main(String[] args)
{
declaration of a constant final float VAT = 0.175f;
declaration of variables float cdPlayer;
float amplifier;
float speakers;
float subTotal;
float tax;
float total;
assign values to the goods purchased cdPlayer = 75.00f;
amplifier = 99.00f;
speakers = 56.00f;
calculate sub total of goods subTotal = cdPlayer+amplifier+speakers;
calculate value added tax
tax = VAT * subTotal;
calculate total cost total = subTotal + tax;
```

```
display shopping bill
Window screen = new Window("Example_2.java","plain","blue",36);
screen.showWindow();
screen.write("HI FI Bulk Discount Stores\n\n");
screen.write("CD Player\t\t"+cdPlayer+"\n");
screen.write("amplifier\t\t"+amplifier+"\n");
screen.write("speakers\t\t"+speakers+"\n\n");
screen.write("sub-total\t\t"+subTotal+"\n");
screen.write("VAT\t\t"+tax+"\n"); screen.write("TOTAL \t\t"+total+"\n");
}
}
```

The results appear as follows.



1.8 Operator Precedence

- Suppose an expression is written as A+B*C-D/E. How would it be evaluated?
- set of rules for the evaluation
- All operators have an associated hierarchy that determines the order of precedence for evaluating expressions

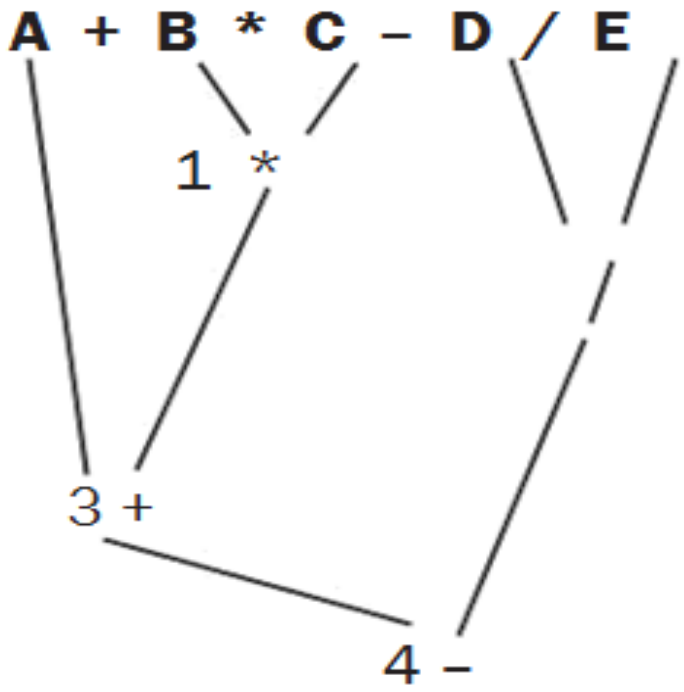
Priority level	Operator	Operand type(s)	Associativity	Operation performed
1	+ –	*arithmetic*	R	unary plus, unary minus
	(type)	*any*	R	cast
2	* / %	*arithmetic*	L	multiplication, division, remainder
3	+ -	*arithmetic*	L	addition, subtraction
13	=	*variable any*	R	assignment

Figure 1.11 Priority levels of operators

A + B * C – D / E

How this expressions can b evaluated?

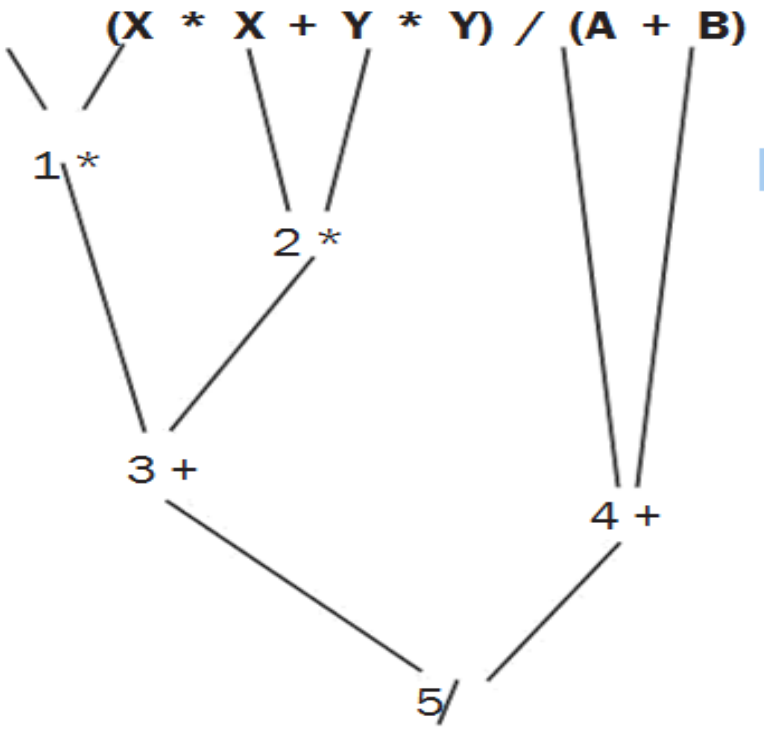
1. (B*C)
2. (D/E)
3. A + (B*C)
4. A + (B*C) - (D/E)



(X * X + Y * Y) / (A + B)

How this expressions can b evaluated?

1. X*X
2. Y*Y
3. ((X*X)+(Y*Y))
4. A + B
5. ((X*X)+(Y*Y))/(A+B)



1.9 Casting

- When operands are of different types, one or more of the operands must be converted to a type that can safely accommodate all values before any arithmetic can be performed.
- Type conversion is performed automatically when the type of the expression on the right-hand side of an assignment can be *safely* promoted to the type of the variable on the left-hand side.

```
long  largeInteger  = 123456789012345;
int   smallInteger  = 987654;

largeInteger = smallInteger;
```

- Note that the assignment smallInteger = largeInteger; is not allowed, since the value of the variable largeInteger cannot be promoted from type long to type int without possible loss of digits.

1.9 Casting (Contd..)

- Type conversion may also be explicit through the use of a cast operation. A cast is an explicit conversion of a value from its current type to another type. The syntax of this operation follows:

For example,

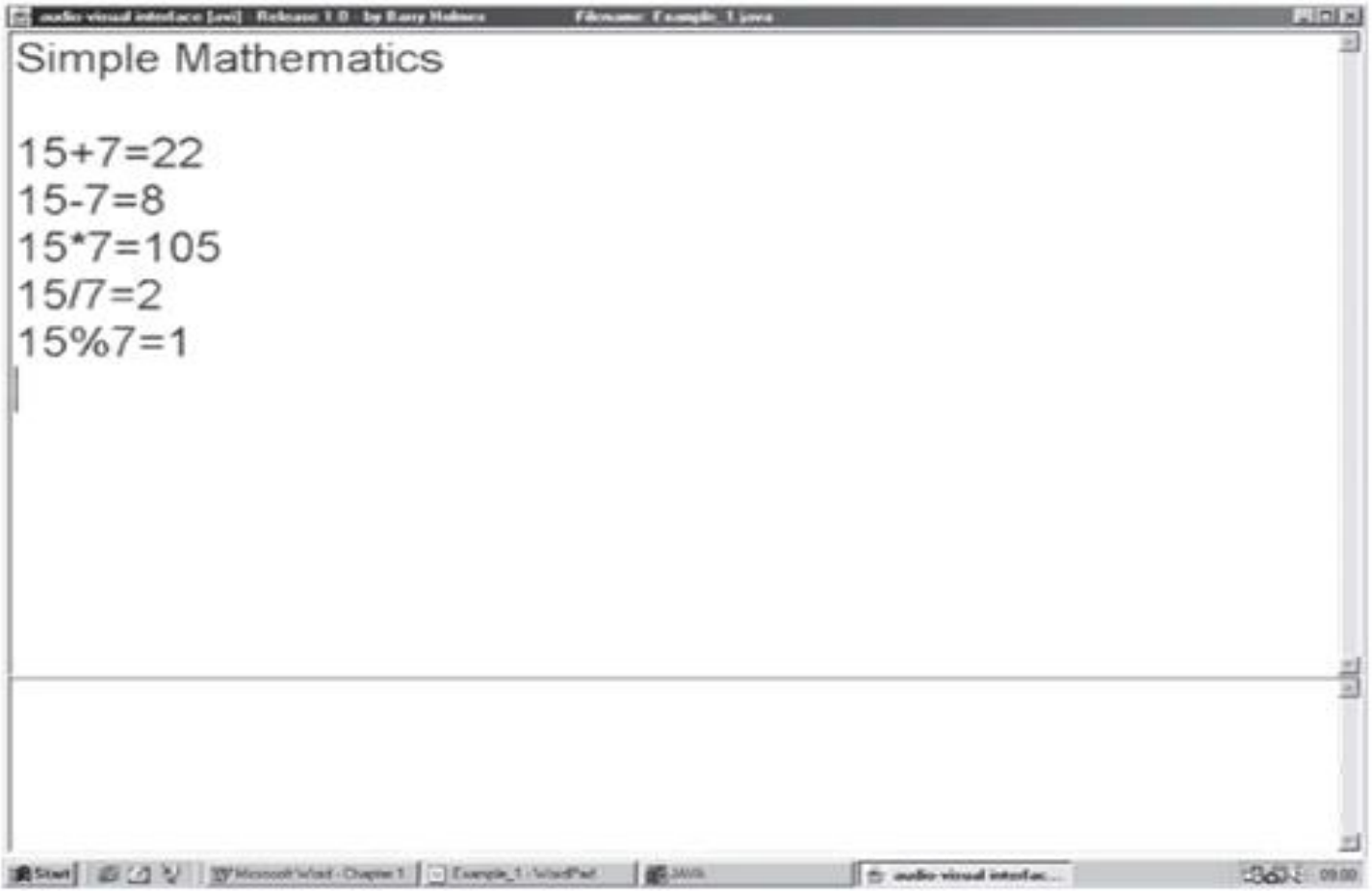
```
float money = 158.05;
int   looseChange = 275;
money = (float) looseChange;
```

program to calculate the sum, difference, product, quotient, and remainder of two numbers

```
import avi.*;

public class Example_3

{

public static void main(String[] args)

{
 declare variables int first;
int second;
int sum;
int difference;
int product;
int quotient;
int remainder;

assign values to variables first = 15;
second = 7;
```

```
perform computations
sum = first+second;
difference = first - second;
product = first*second;
quotient = first/second;
remainder = first%second;

output results of calculations

Window screen = new Window("Example_1.java","plain","blue",36);
screen.showWindow();
screen.write("Simple Mathematics\n\n");
screen.write(first+"+"+second+"="+sum+"\n");
screen.write(first+"-"+second+"="+difference+"\n");
screen.write(first+"*"+second+"="+product+"\n");
screen.write(first+"/"+second+"="+quotient+"\n");
screen.write(first+"%"+second+"="+remainder+"\n");
}
}
```

The following screen shot shows the results from running this program.



Introduction to objects

objects

- an object encapsulates data and a set of operations that access and manipulate the data.
- For example, a program to help manage a bank might maintain thousands of BankAccount objects, each holding data (owner name,balance,interest rate) about a particular bank account and each allowing a set of operations (deposit, withdraw, balance, printStatement) that affect the data or return information about the data.

objects

- Objects are created from templates, called classes.
- A class may define both the type of data and the operations that can be performed on the data.
- These operations are segments of program code, known as methods.
- Once a class is defined, it can be used to create, or instantiate, many objects.
- Sometimes we refer to these objects as instances of the class.

Java API

- In the Java API, classes that are related to each other are grouped together into packages.
- In fact the number of packages increases with every new release.
- A package is a convenient way of grouping together many different classes that have a common purpose.
- The Java API is composed of many packages.
- A package consists of a set of related classes.
- Classes act as templates for objects. They define both data and operations (methods).
- Specific objects are instantiated from a class.
- Objects and primitive data values are the two kinds of information manipulated by Java programs.

The String Class

- A string is a group of characters that are stored as consecutive items in the memory of a computer, with each character being represented by a 16-bit Unicode.

- A string literal in Java is delimited by double quotes.

- For example the string literal ABC is written as “ABC”, or by using the Unicodes as “\u0041\u0042\u0043”.

Declaring Objects

- How can we declare a string object?
- By using the name of the class String in the same way as you would use the names of any of the primitive types.
- A class may be thought of as a data type.
- For example, you can declare a *String* object called *alphabet* as follows:

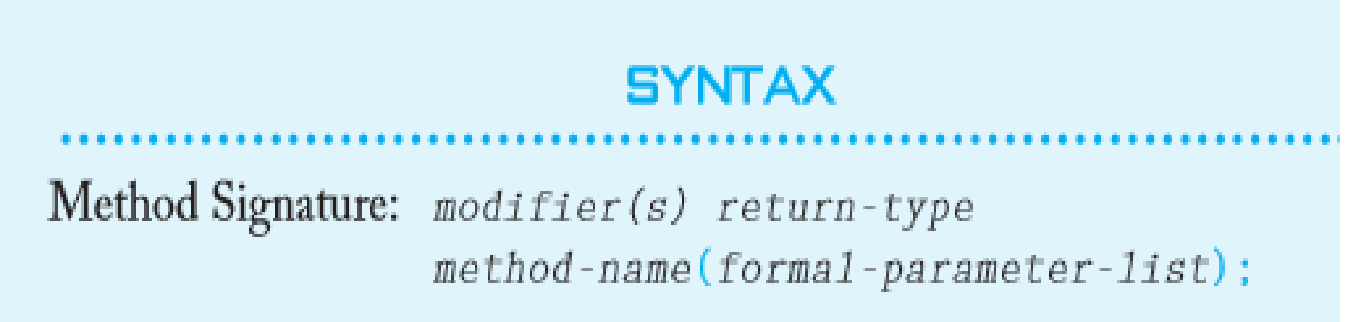
  String alphabet;

- The declaration on its own is not much use, since the memory location alphabet does not refer to any data.
- It simply reserves a location in memory for a reference to the object and has designated the contents of this location as null.

Methods and Parameters

- a class may define both data and segments of program code, known as methods, to operate upon the data contained in the class.
- A method’s signature is the first line of a method, terminated by a semicolon.
- The purpose of a signature is to uniquely identify a method in terms of its name and formal parameter list.

- The syntax of a signature is:



- modifier(s)—usually indicates the visibility of the method, i.e., where it can be activated from.
- return-type—the return type specifies the data type of the value that is returned by the method.
- method-name—an identifier that defines the name of the method. formal-parameter-list—declares the data variables that are passed to and used by the method

The String class

Method signatures are used in the documentation of a class. Consider the following partial listing taken from the `String` class.

```
public final class String ...
{
   // constructors
   public String();
   public String(String value);
   .
   .

   // instance methods
   public String concat(String str);
   public int length();
   public String replace(char oldChar, char newChar);
   public String toUpperCase();
   .
   .
}
```

Constructors

- the methods that have the same name as the class.
- These are special methods known as constructors, and their purpose is to initialize data of the type String to specified values.
- A constructor does not return a value.
- Recall that we declared a String object called alphabet as follows:

String alphabet;

- Once it is declared, we can initialize alphabet by using either of the constructors.
- This initialization is known as creating an instance of the class or creating an object.

Creation of object

SYNTAX

Instantiation:

```
object-name = new class-constructor();
object-name = new class-constructor(argument-list);
```

where `argument-list` consists of one or more values used by the constructor to initialize the data of the object.

Examples of creating the object `alphabet` follow.

```
alphabet = new String();
```

The above uses the first `String` constructor with no arguments.

```
alphabet = new String("abcdefghijklmnopqrstuvwxyz");
```

- Since the declaration and initialization of objects are so frequently performed one right after the other, the Java language provides a way to do both together in a single statement.
- The following will both declare and instantiate the String object alphabet:

```
String alphabet = new String("abcdefghijklmnopqrstuvwxyz");
```

Primitive types stored by value

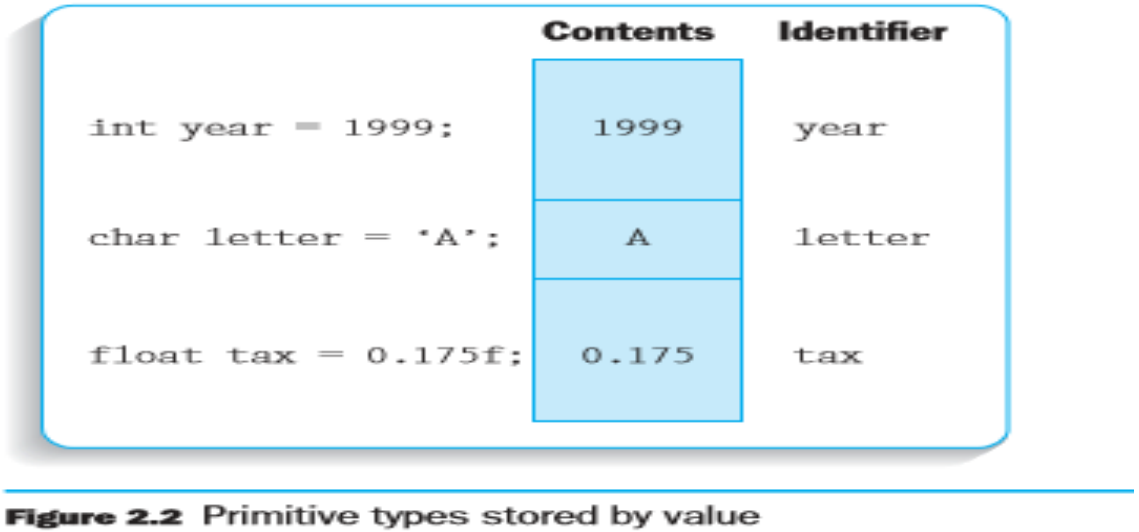


Figure 2.2 Primitive types stored by value

In Figure 2.2,the values of the identifiers year, letter, and tax are stored at the memory locations depicted by the names of the identifiers.

Hence, the primitive data is stored by value.

An object is stored by reference

- when an identifier of the type String is initialized, the value of the string is not stored at the memory location depicted by the identifier, but it is stored in a different location pointed at or referenced by the identifier.
- The object alphabet is stored by reference.
- The purpose of the reserved word new is to allocate a new memory storage area for holding the value of the string.

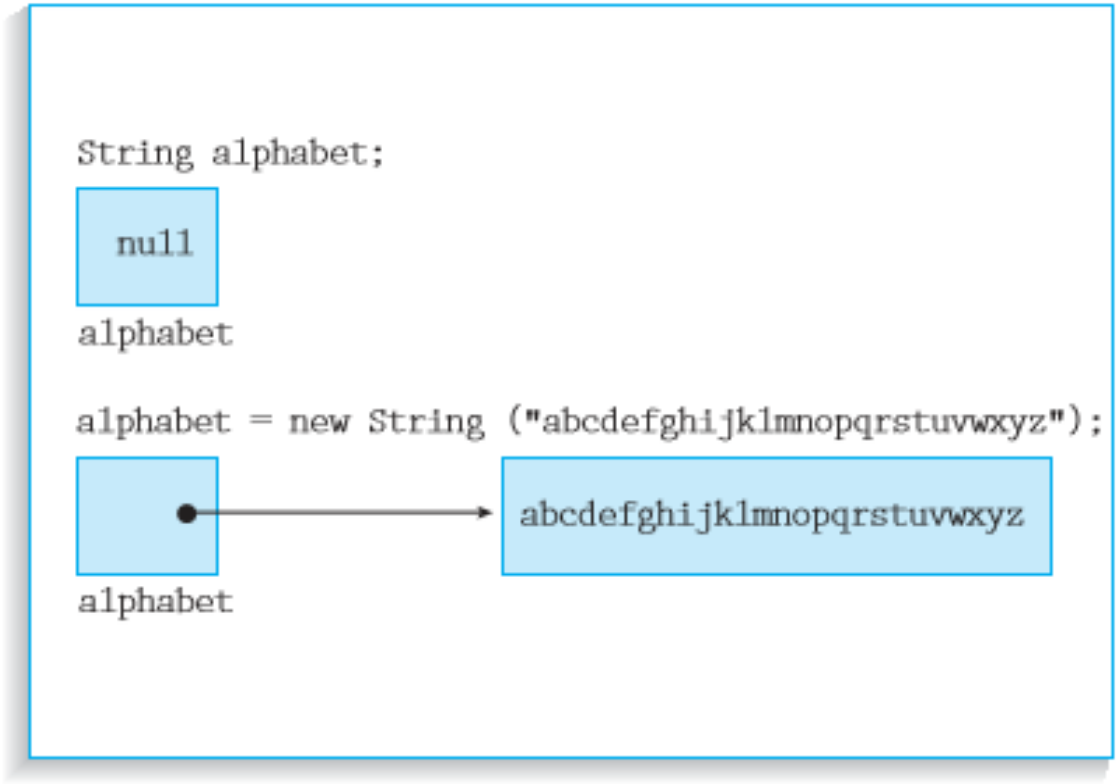


Figure 2.3 An object is stored by reference

String Assignment

If you wish to assign one string to another, then the assignment does not pro-vide a copy of the value but merely a copy of the *reference* to the value.

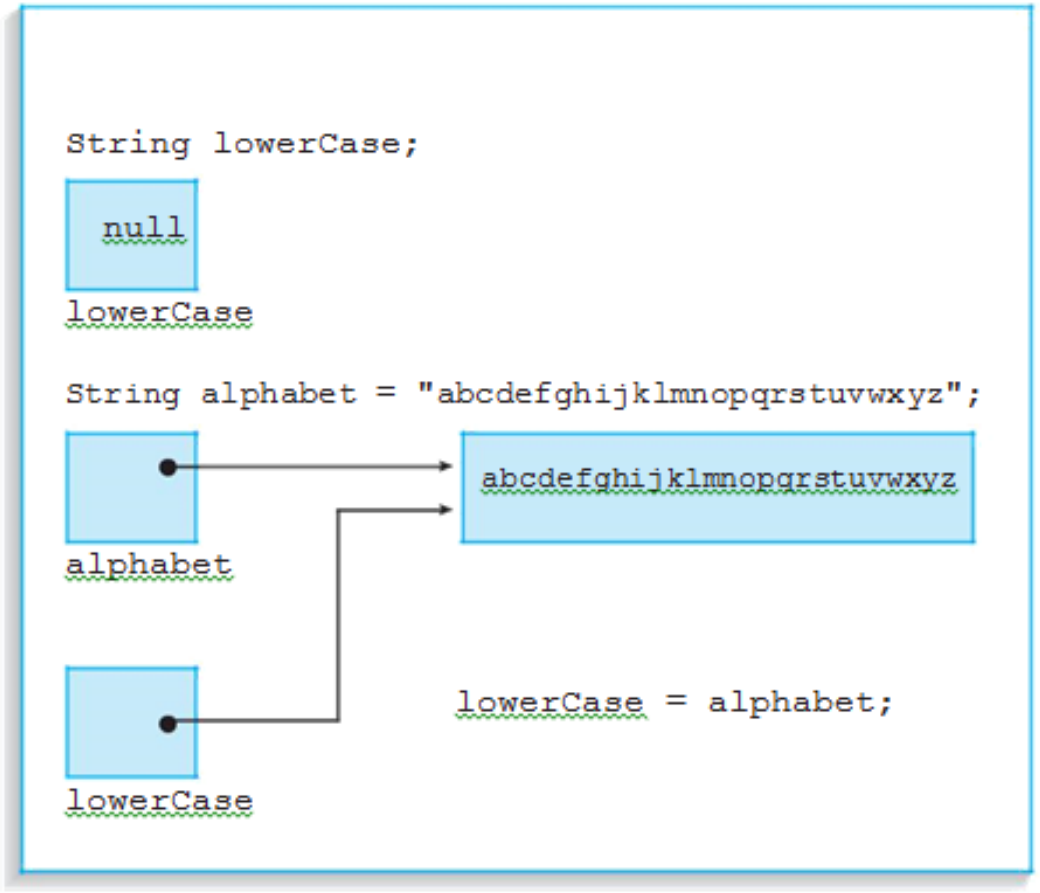


Figure 2.4 Assignment of strings

Instance Methods

From the partial listing of the class String you can see a group of methods that appear to describe the characteristics and operations you might associate with an object such as a string of characters.

For example,

- the identifier **length** suggests that it returns the characteristic of the number of characters in a string;
- the identifier **concat** suggests the operation of concatenation or appending of one string after another;
- the identifier **replace** suggests the operation of replacing every occurrence of one character with another;
- and the identifier **toUpperCase** suggests the operation of converting the characters in the string to the uppercase letters if appropriate.

SYNTAX

Passing a Message to an Object by an Instance Method:

```
object.method-name();
object.method-name(argument-list);
```

The statement **alphabet.length()** will return the length of the alphabet string as 26.

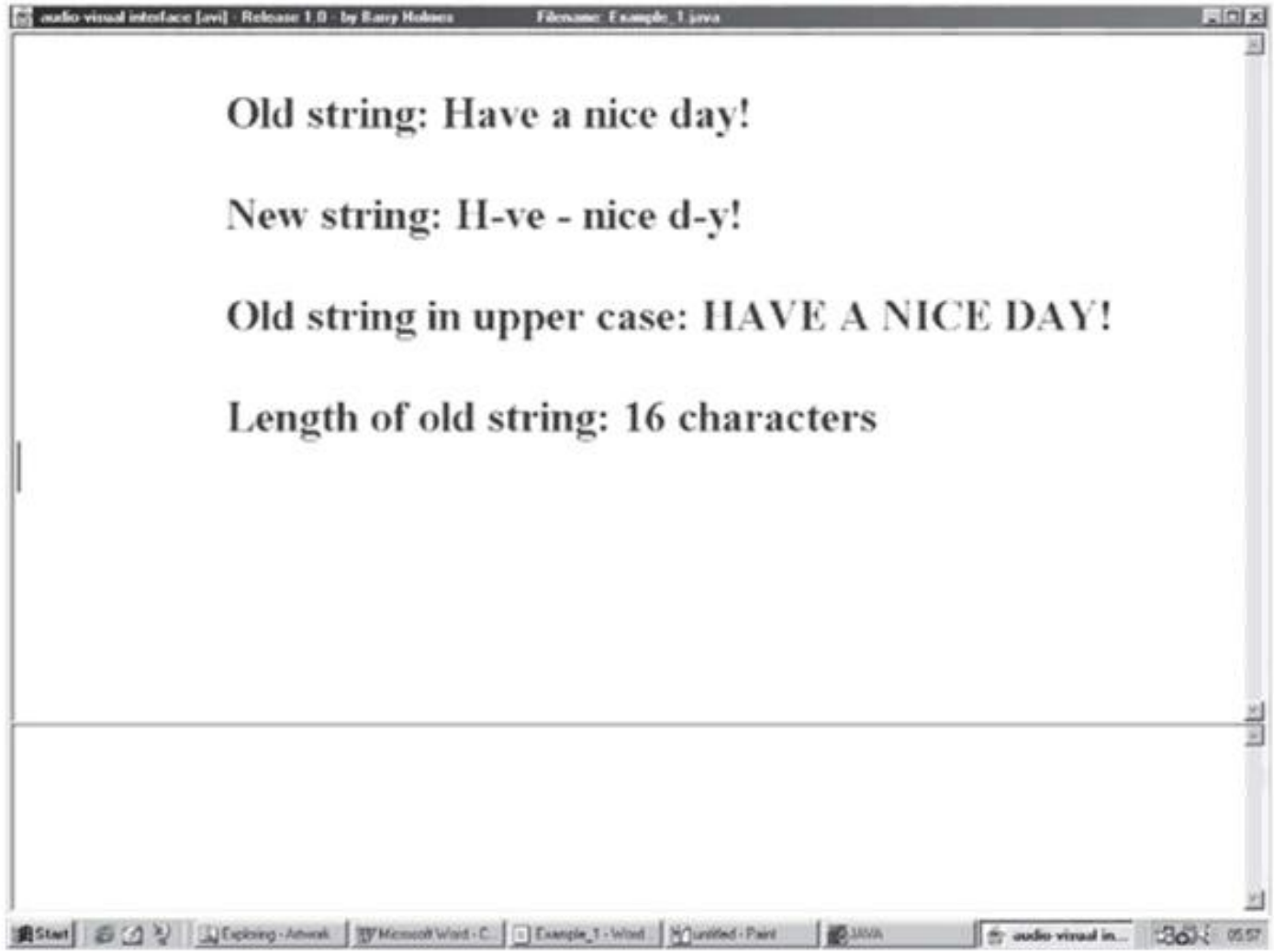
The statement **alphabet.toUpperCase()** will change every character of the alphabet string to uppercase letters:

"ABCDEFGHIJKLMNOPQRSTUVWXYZ".

program to demonstrate the String class and some of its instance methods import avi.*;

```
class Example_1
{
public static void main(String[] args)
{
String oldString = "Have a nice day!";
String newString = oldString.replace('a','-');
String capitalString = oldString.toUpperCase();
int lengthOfString = oldString.length();

Window screen = new Window("Example_1.java","bold","blue",36); screen.showWindow();
screen.write("\n\tOld string: " + oldString + "\n");
screen.write("\n\tNew string: " + newString + "\n");
screen.write("\n\tOld string in upper case: " + capitalString + "\n");
screen.write("\n\tLength of old string: " + lengthOfString + " characters\n");
}
}
```

The following is the resulting screenshot.



Import List

- The import statement makes Java classes available to the program.
- You can specify each class in the import statement, for example avi.DialogBox and avi.Window, but it is a lot simpler to use an asterisk as a wildcard to make all the classes of the package avi available;

hence the statement avi.*

The AVI Package

- Figure indicates the names of the classes within the avi package that are available for public use.
- This figure uses the Universal Modeling Language (UML) notation for describing a package.

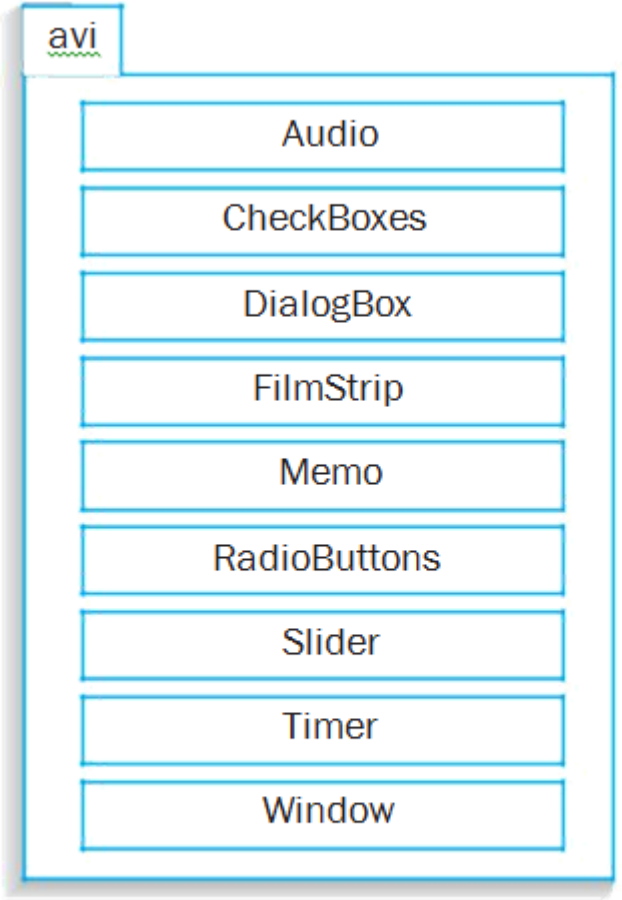


Figure 2.5 UML representation of the `avi` package

The Window Class

- The Window class is used for *output* and is analogous to a pane of glass in a win-dow frame.
- You can write on the glass or fix objects, such as posters, to the glass.
- By creating a Window object, you are creating a container for both a writing area and for displaying other graphical objects.

```
public class Window
{
        // first constructor
        public Window(String filename);

        // second constructor
        public Window(String filename,
                      String style,
                      String color,
                      int    fontSize);

        public void showWindow();
        public void clearTextArea();
        public void closeWindowAndExit();
        public int getWidth();
        public int getHeight();
        public void write(String datum);
        public void write(char datum);
        public void write(int datum);
        public void write(long datum);
        public void write(float datum);
        public void write(double datum);
}
```

The Window Class (Contd…)

filename—is normally the name of the main application file associated with the program. This parameter is for documentation purposes only.

style—is either "PLAIN", "BOLD", "ITALIC" or "BOLD+ITALIC". You may use either uppercase or lowercase characters. An empty string "" will default to a BOLD+ITALIC style.

color—is either "red", "blue", or "black". You may use either uppercase or lowercase characters. An empty string "" will default to the color black.

fontSize—is a positive integer that represents the point size of the output text. Values less than 10 default to a point size of 10. A font size of 72 points will produce uppercase characters approximately 1 inch in height.

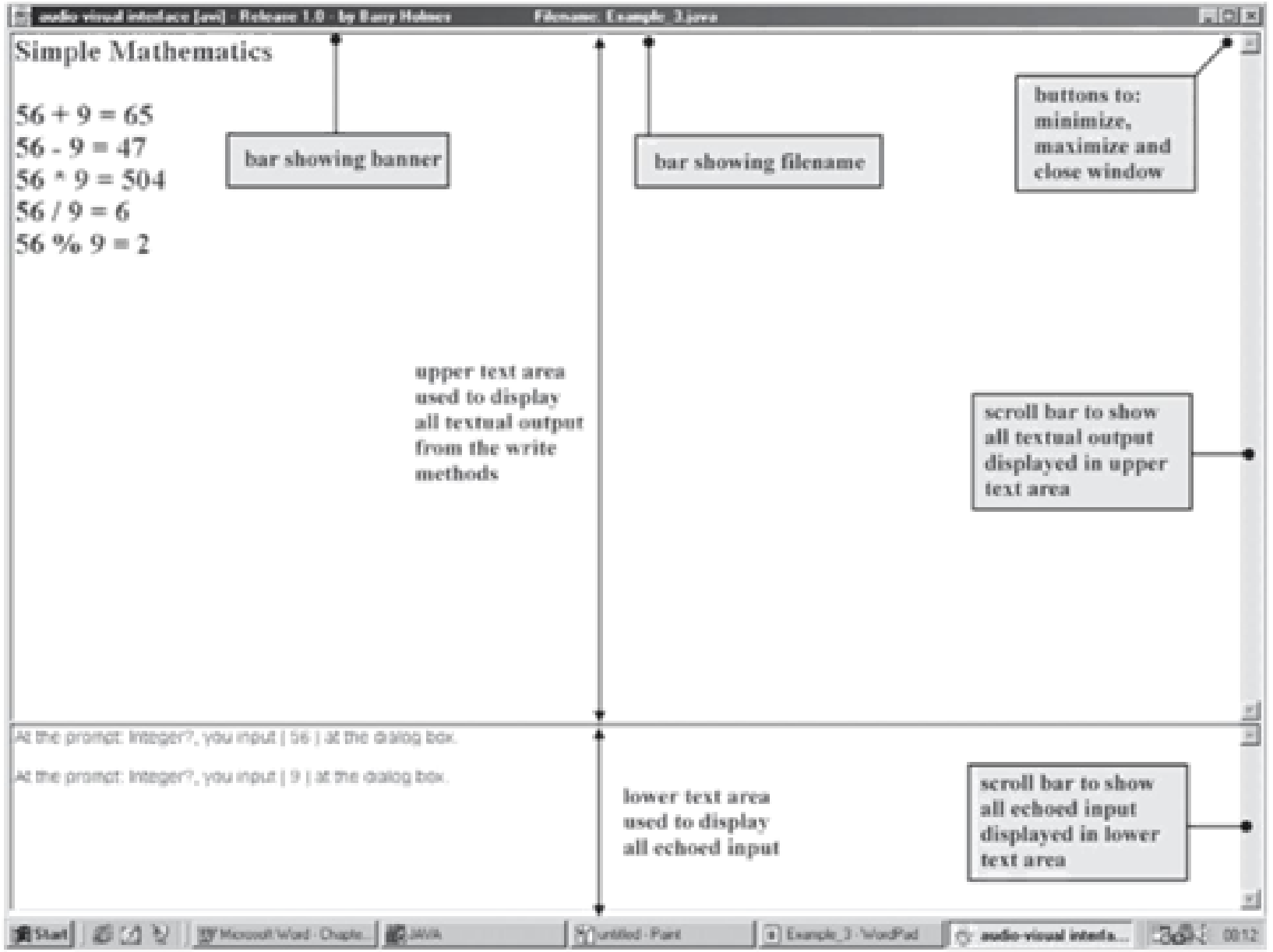


Figure 2.6 Format of a `Window` object

Input to A Dialog Box

- The DialogBox class is used for the input of any string value via the keyboard.
- The constructor creates a DialogBox object that appears on the Window object.
- The dialog box is modal, implying that no other interaction either with the win-dow pane or with other objects on the pane is possible until data has been input at the dialog box and the box has been closed.

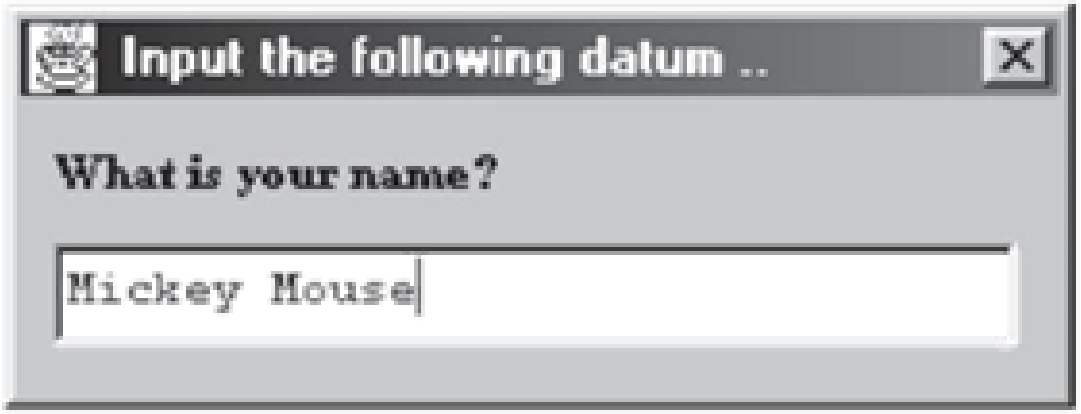


Figure 2.7 An example of a dialog box

Input to A Dialog Box

```
public class DialogBox
{
      public DialogBox(Window parent,
                       String prompt);

      public void showDialogBox();
      public String getString();
      public char getChar();
      public int getInteger();
      public long getLongInteger();
      public float getFloat();
      public double getDouble();
}
```

To create a DialogBox object you must use the class constructor.

parent—a Window type that specifies the container onto which to place the dialog box.

prompt—a string, used as a cue to prompt for input. For example, in Figure 2.7 the prompt is "What is your name?".

program to input your name and display a welcome message on the screen

```
import avi.*;
class Example_3
{
public static void main(String[] args)
{
declare name as a string
String name;

create a window object screen
Window screen = new Window("Example_3.java","bold","blue",48);
screen.showWindow();

create a dialog box object for user input
DialogBox inputName = new DialogBox(screen,"What is your name?");
inputName.showDialogBox();

get the name from the dialog box
name = inputName.getString();

display a welcome message on the screen
screen.write("\n\n\n Hello "+name+"!");
screen.write("\n\n Welcome to");
screen.write("\n Object-oriented Programming with Java.");
}
}
```

Converting Strings to Numbers

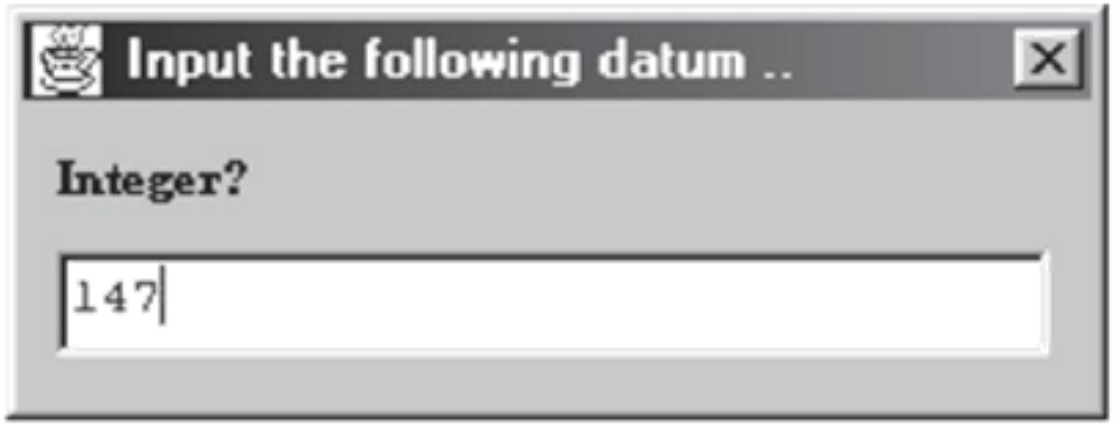


Figure 2.8 Input of a number into a dialog box

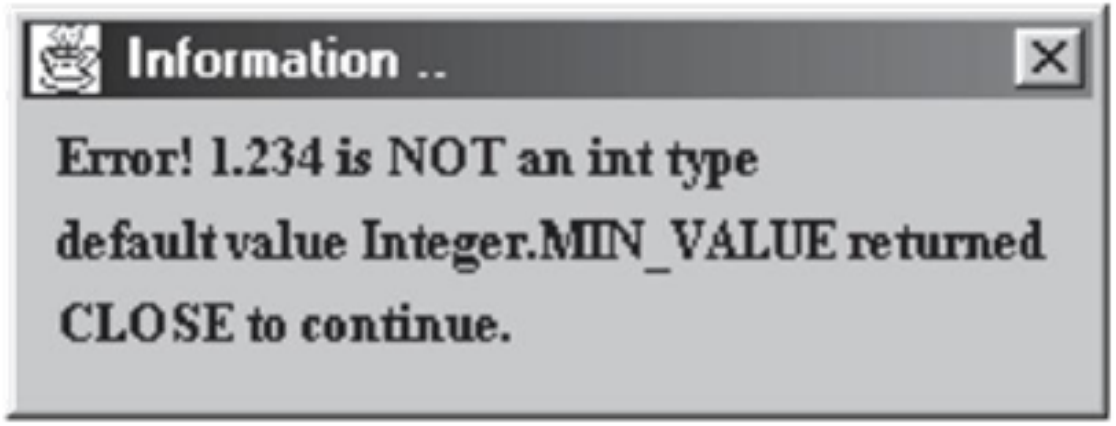


Figure 2.9 An error message issued when data is not in the correct format

Program to input two integer operands and perform the arithmetic operations of +, - *, /, and % upon them

1. declare two variables of type integer
2. create a window object screen
3. display a heading on the screen
4. create a dialog box object to input numbers
5. show dialog box twice to input two integers
6. display the results of calculations

```
import avi.*;
class Example_4
{
public static void main(String[] args)
{
int first, second;

Window screen = new Window("Example_4.java","bold","blue",24);
screen.showWindow();

screen.write("Simple Mathematics\n\n");

DialogBox inputNumber = new DialogBox(screen,"Integer?");

inputNumber.showDialogBox();
first = inputNumber.getInteger();
inputNumber.showDialogBox();
second = inputNumber.getInteger();

screen.write(first+" - "+second+" = "+(first+second)+"\n");
screen.write(first+" - "+second+" = "+(first-second)+"\n");
screen.write(first+" * "+second+" = "+(first*second)+"\n");
screen.write(first+" / "+second+" = "+(first/second)+"\n");
screen.write(first+" % "+second+" = "+(first%second)+"\n");
}
}
```

Command Line Arguments

The command line provides another means of inputting *string* data to a program at the point at which you issue the command to run the program.

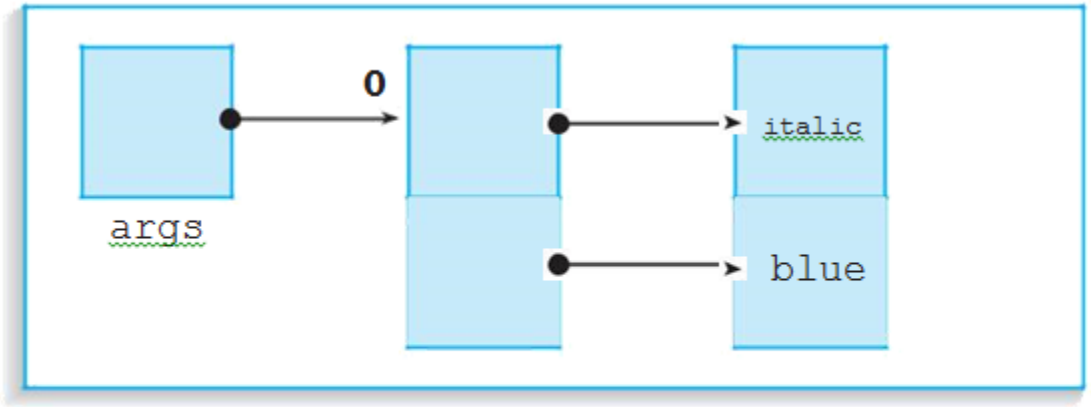


Figure 2.11 An illustration of the `main` method formal parameter `args`

program to use command line parameters to specify the style and color of the text on the window at run time, and write the text literal "HELLO WORLD" centrally on the screen

```
import avi.*;
class Example_5
{
public static void main(String[] args)
{
create a Window object screen Window screen = new Window("Example_5.java",args[0],args[1],72);
screen.showWindow();
screen.write("\n\n\n HELLO WORLD");
}
}
```

The program is executed using the command line:

```
java Example_5 italic blue
```

ERRORS

1. Syntax Errors - the misuse of the syntax (grammar) of the language.

For Example : type a lowercase letter when an uppercase letter was required, omit a semicolon at the end of a statement, incorrectly type the wrong number of arguments in a constructor or method, and so on.

2. Run Time Errors - error you may encounter occurs when the program compiles correctly, yet fails during program execution.

For Example : when it is required to input three values, if you do not input the three command-line parameters, you will encounter run time errors

3. Logical Errors - when the program compiles correctly, yet does not perform in the manner expected.

For example : If you coded a minus sign somewhere you meant to code a plus sign, your program would compile and run to completion but would probably give incorrect output.