Java Programming

Session 1:

- Java Language and its features
- Java vs. C++
- Primitive Data Types and Operators
- Control Statements
- Classes and Objects
- Inheritance

Java Language and its features

Introduction to Java

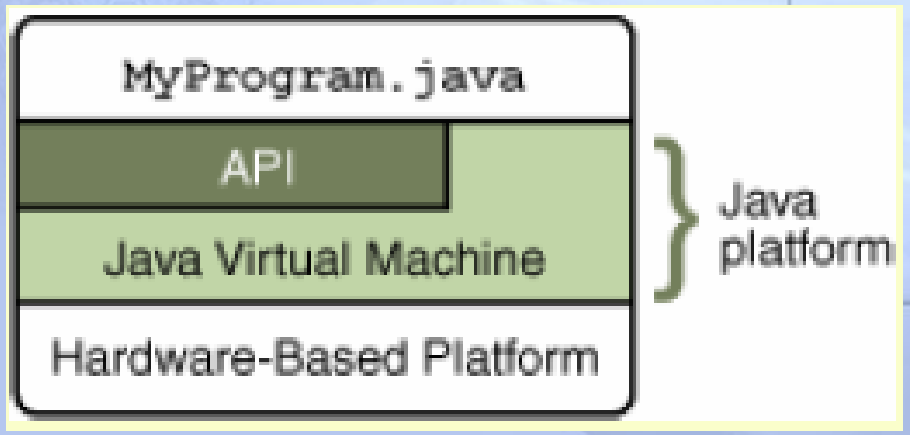
- Java is an object-oriented programming language with a built-in application programming interface (API) that can handle graphics and user interfaces and that can be used to create applications or applets.
- Because of its rich set of API's, and its platform independence, Java can also be thought of as a platform in itself. Java also has standard libraries for doing mathematics.
- Much of the syntax of Java is the same as C and C++. One major difference is that Java does not have pointers.
- However, the biggest difference is that you must write object oriented code in Java.

Java's Features

- The Java programming language is a high-level language that can be characterized by all of the following features:
  - ✓ **Simple**
  - ✓ **Object-oriented (interpreted)**
  - ✓ **Distributed**
  - ✓ **Interpreted**
  - ✓ **Robust**
  - ✓ **Secure**
  - ✓ **Architecture neutral**
  - ✓ **Portable**
  - ✓ **High performance**
  - ✓ **Multithreaded**
  - ✓ **Dynamic**

The platform

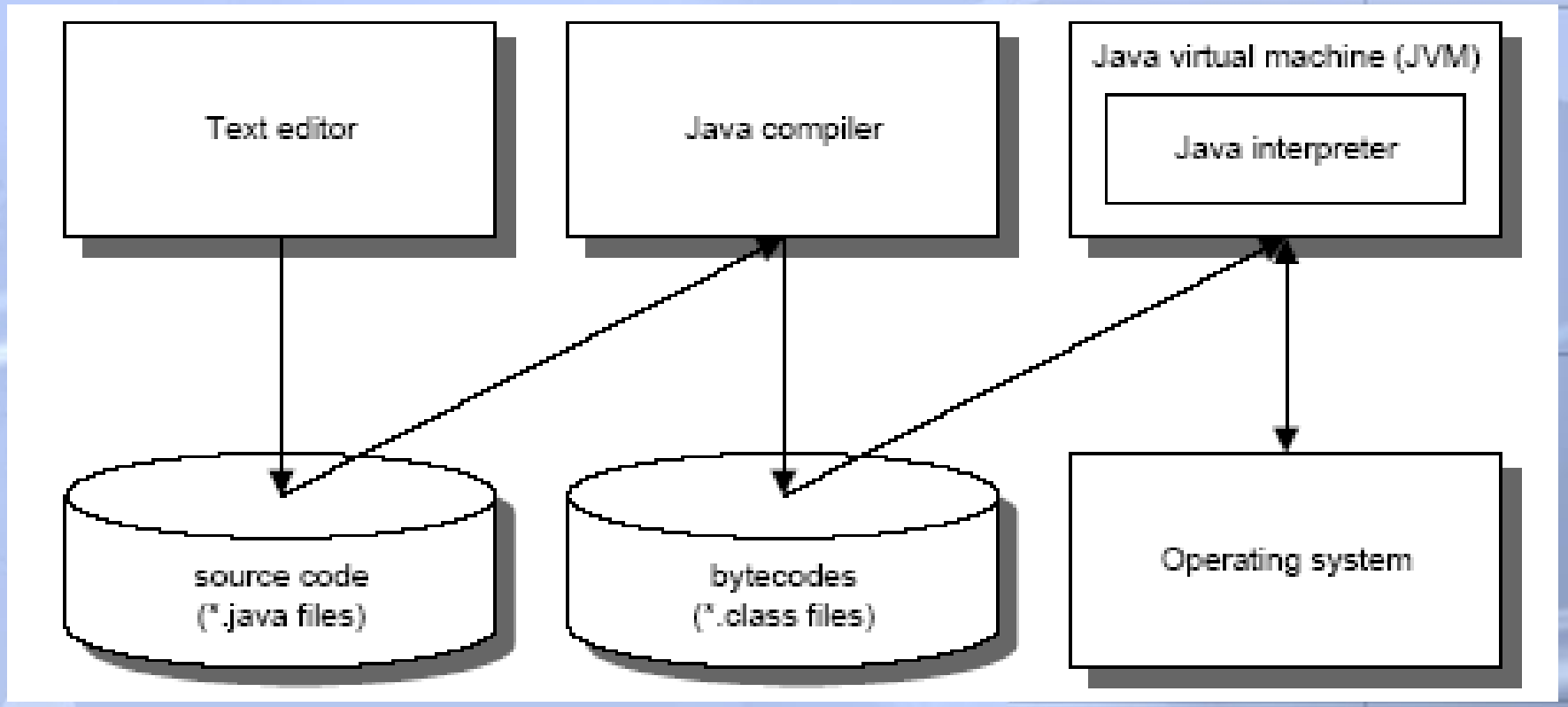
- A *platform is the hardware or software* environment in which a program runs
  - Microsoft Windows, Linux, Solaris OS, and Mac OS
- The Java platform has two components:
  - The *Java Virtual Machine*
  - The *Java Application Programming Interface (API)*



Java Environment

- Programs come in two kinds
  - **Applications**
    - Unrestricted access to system resources
    - Interface can be graphical, textual or neither
  - **Applets**
    - Restricted access to system resources
    - Interface is embedded in some graphical wrapper
      - Browser
      - Appletviewer

Java Compilation & Execution



Benefits of Java

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program written in C++.
- **Write better code:** The Java programming language encourages good coding practices, and automatic garbage collection helps you avoid memory leaks.
- **Develop programs more quickly:** The Java programming language is simpler than C++, and as such, your development time could be up to twice as fast when writing in it. Your programs will also require fewer lines of code.

Benefits of Java

- **Avoid platform dependencies:** You can keep your program portable by avoiding the use of libraries written in other languages.

- **Write once, run anywhere:** Because applications written in the Java programming language are compiled into machine independent bytecodes, they run consistently on any Java platform.

- **Distribute software more easily:** With Java Web Start software, users will be able to launch your applications with a single click of the mouse. An automatic version check at startup ensures that users are always up to date with the latest version of your software. If an update is available, the Java Web Start software will automatically update their installation.

Java vs. C++

- C++ is not cross-platform
- C++ is not pure object-oriented.
  - It can have functions outside a class
- Relies heavily on pointers.
  - Memory errors can occur
- C++ creates an executable that is platform dependent.
  - Java programs are compiled to platform independent class files.
- C++ is faster than Java

Primitives Data Types and Operators

Data Types

- Java supports two kinds of types of values
  - **objects**, and
  - values of **primitive** data types

- variables store
  - either references to objects (address in memory)
  - or directly primitive values (bit patterns)
    - bit patterns is interpreted according to the primitive type

Primitive Data Types

- primitive types defined in Java
  - byte, short, int, long (Integral)
    - integer numbers only different precision (8, 16, 32, 64 bits)
  - float, double (Floating)
    - floating point numbers only different precision (32, 64 bits)
  - char (Textual)
    - characters (letters, digits, special characters)
  - boolean (Logical)
    - logical values true or false

Primitive Data Types

- Integer types
  - byte (1 byte signed)
  - short (2 bytes signed)
  - int (4 bytes signed)
  - long (8 bytes signed) use suffix L (eg 1000000000L)
- Floating-point types
  - float (4 bytes) use suffix F (eg 1.28F)
  - double(8 bytes)
- char
  - Two-byte unicode (16 bits, but stored in one 32-bit word)
  - Assignment with ‘ ’ e.g. char c = ‘h’;
- boolean
  - true or false e.g. boolean x = true;

if (x){…};

Data Type's Default Values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Identifier

- Identifier is a word used by a programmer to name a variable, method, class.
- ✓ Must begin with **a letter (a-z), a dollar sign ($), or an underscore (_)**
- ✓ Subsequent characters may be **letters, dollar signs, underscores, or digits**.
- ✘ Don't use keywords and reserved words (e.g., class, static, int, etc..)
- The following are legal identifiers:
  - ☑ Dummy
  - ☑ box$
  - ☑ A37
  - ☑ post_code
- The following are not legal identifiers:
  - ✘ rain@noon
  - ✘ 2by4
  - ✘ post code

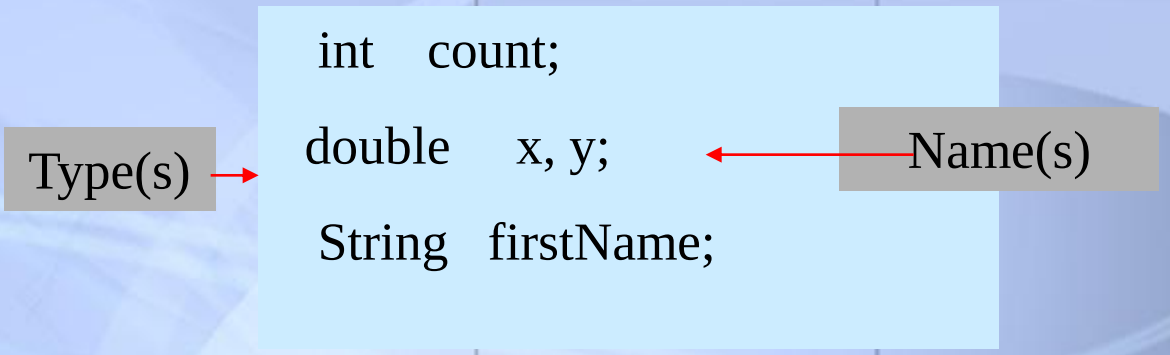
test	✓	
Bigtest	✓	
$test	✓	
1test	✗	Start with digit.
!test	✗	Must start with letter, $ or _
test test	✗	Can't use space

Identifier(cont'd)

- Case is significant:
  - identifiers ***Abc***, ***aBc*** and ***abC*** are different
- Identifiers cannot contain special operation symbols like +, -, *, /, &, %, ^, etc.
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.
- Names of variables usually start with a lowercase letter.

Variables (cont'd)

- A variable must be declared before it can be used:



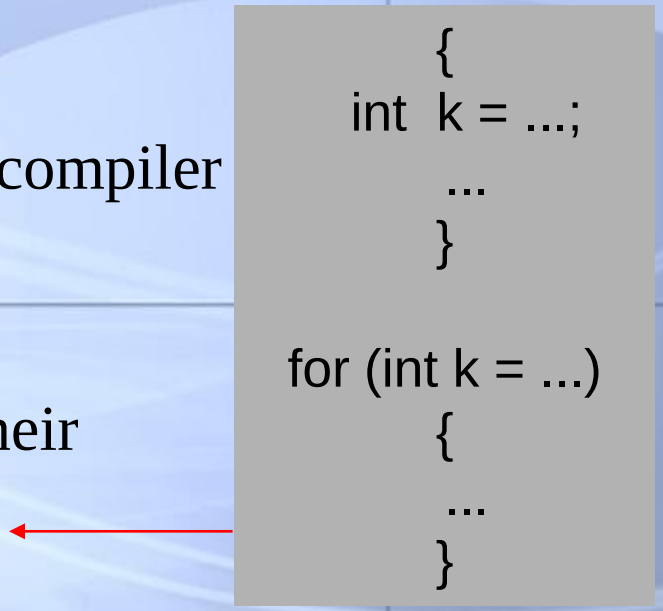
- Variables can be defined just before their usage (unlike C)
- The assignment operator = sets the variable's value.
- A variable can be initialized in its declaration:

```
count = 5;
x = 0.0;
firstName = args[0];
```

```
int count = 5;
String firstName = args[0];
```

Variables: Scope

- Each variable has a scope — the area in the source code where it is “visible.”
- If you use a variable outside its scope, the compiler reports a syntax error.
- Variables can have the same name when their scopes do not overlap.



Constants

- Constants are similar to variables except that they hold a fixed value. They are also called “READ ONLY” variables.

- Constants are declared with the reserved word “final”.

```
final int MAX_LENGTH = 420;
final double PI = 3.1428;
```

- By convention , upper case letters are used for defining constants.

Expressions

- Variables and constants are used in expressions to define new values and to modify variables.
- Expressions involve the use of *literals*, *variables*, and *operators*

Literals:

A literal is any "constant" value that can be used in an assignment or other expression.

- null, true, false, integer value, floating point value,
- Character: a character is defined as an individual symbol enclosed in single quotes, 'a', '!', '\n'….
- String literal: A String literal is a sequence of characters enclosed in double quotes, "EE205 Students", "KAIST".

Operators

- primitive types support operators
  - typically binary, but also unary and even ternary
  - used in expressions
- arithmetic operators
- assignment operators
- Increment /decrement operators
- relational operators
- logical operators

Arithmetic Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- % Modulus
- ++ Increment
- -- Decrement
- += Operator Assignments

Arithmetic Operators

- -=
- *=
- /=
- %=
  - numeric operands, result is the more precise number type
    - byte, short, int, long, float, double
    - +, -, *, /, % (modulo)
    - +, - (sign change, unary)

Arithmetic Operator

- The precedence of operators and parentheses is the same as in algebra
- Get the remainder with % (pronounced "modulo")
- m % n means the remainder when m is divided by n

  5 % 2 yields 1 (the remainder of the division)
- % has the same rank as / and *
- Same-rank binary operators are performed in order from left to right

- / is the division **operator**
- If both arguments are integers, the result is an integer. The remainder is discarded .

  5/2 yields an integer 2.

  5.0/2 yields a double value 2.5

Increment & Decrement Operator

- Prefix increment ++ e.g. ++i

Increase i by 1, then use the new value of i to evaluate the expression that i resides

- Postfix increment ++ e.g. i++

Use the current value of i to evaluate the expression that i resides, then increase i by 1

- Prefix decrement -- e.g. --i

Decrease i by 1, then use the new value of i to evaluate the expression that i resides

- Postfix decrement -- e.g. i--

Use the current value of i to evaluate the expression that i resides, then decrease i by 1

- int m = 7; int n = 7;

  int a = 2* ++m; // a =16

  int b = 2* n++; // b = 14

Compound Assignment Operator

- most operators can be combined with an assignment
  - meaning: x <op>= y; means x = x <op> y

Operator	Example	Meaning
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Relational Operators

- == Equal to (works also with objects!)
- != Not equal to (works also with objects!)
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
  - primitive types operands, boolean result

    3 == 7; 3 != 7
- Apply to numbers or chars:

```
if ( count1 <= count2 ) ...
if ( sum != 0 ) ...
if ( letter == 'Y' ) ...
```

Bitwise Operators

- ~ Bitwise unary NOT
- & Bitwise AND
- | Bitwise OR
- ^ Bitwise exclusive OR
- >> Shift right
- >>> Shift right zero fill
- << Shift left
- &= Bitwise & operator

Logical Operators

– boolean operands, boolean result

- && Short-circuit AND

(short circuit, i.e. second operand is evaluated only if necessary)

- || Short-circuit OR
- ! Logical unary NOT
- == Equal to
- != Not equal to

Conditional (short-circuit) AND (&&)

- If (I > 5 && j > 5)

  System.out.println("Both i and j are greater than 5.");

expression1	expression2	&&
false	false	false
false	true	false
true	false	false
true	true	true

Conditional (short-circuit) OR (||)

- If (I > 5 || j > 5)
  System.out.println("Either i is greater than 5 or j is greater than 5");

Expression 1	Expression 2	||
false	false	false
true	false	true
false	true	true
true	true	true

Ternary Operator ?:

- ternary operator <op1> ? <op2> : <op3>
  - <op1> must be boolean
  - result is <op2> if <op1> is true
  - result is <op3> if <op1> is false
  - e.g.,

```
int minimum = i < j ? i : j;
x < y ? x: y;          // if x < y is true, x else y
```

Operator Precedence

Operator Type	Operator	Associativity
Unary	[]. (Params) E++ E--	Right to Left
Unary	Unary operators: -E !E ~E ++E –E	Right to Left
Object creation	new (type)E	Right to Left
Arithmetic	* / %	Left to Right
Arithmetic	+ -	Left to Right
Bitwise	>> << >>>	Left to Right
Relational	<> <= >=	Left to Right
Relational	== !=	Left to Right
Bitwise	&	Left to Right
Bitwise	^	Left to Right
Bitwise	|	Left to Right
Logical	&&	Left to Right
Logical	||	Left to Right
Conditional	?:	Left to Right
Assignment	= += -= *= /= >>= <<= &= ^= |=	Right to Left

Control Statements

Control Statements

- **if** statement
  - Nested if's
  - else if
  - block vs. single statement
- **switch** statement
  - byte, short, int, char
  - more efficient
  - default optional
- Loops
  - **while()**{} :takes boolean
  - **do{}while()**: evaluated after the loop
  - **for** (init; condition; iteration){}: multiple inits and iterations,declared variables only exist in loop, no need for init, condition, or iteration

Jump Statements

- **break**
  - exits loop
  - does not work when inside switch
- **break *label***
  - usually in nested loops
  - jumps to block marked with label
- **continue**
  - skips to next iteration of loop

switch Statements

```
switch (year)
{
  case 7:  annualInterestRate = 7.25;
           break;
  case 15: annualInterestRate = 8.50;
           break;
  case 30: annualInterestRate = 9.0;
           break;
  default: System.out.println(
   "Wrong number of years, enter 7, 15, or 30");
}
```

for-each loop

```
for (variable: collection )
{
   statement1;
   statement2;
   ...
   statementN;
}
```

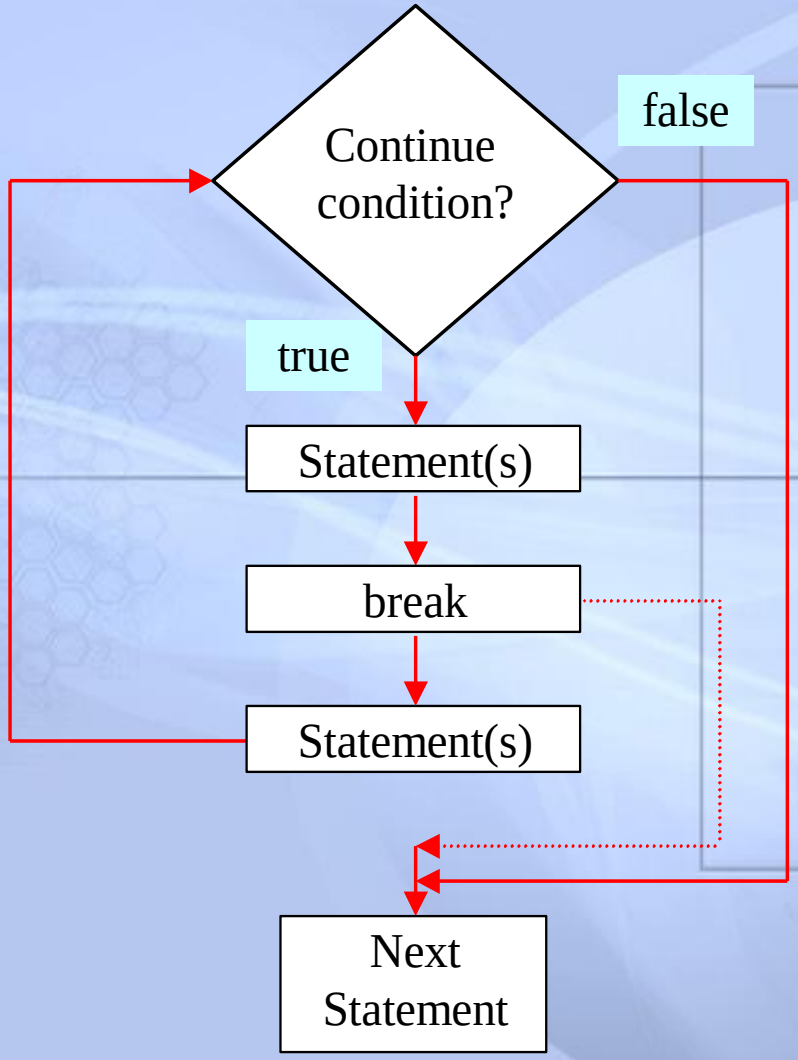
```
int num[]={1,2,3,4,5};

For (int n: num)
{
        System.out.println(n);
}
```

break in Loops

- break in a loop instructs the program to immediately quit the current iteration and go to the first statement following the loop.

- A break must be inside an if or an else, otherwise the code after it in the body of the loop will be unreachable.

The break Keyword

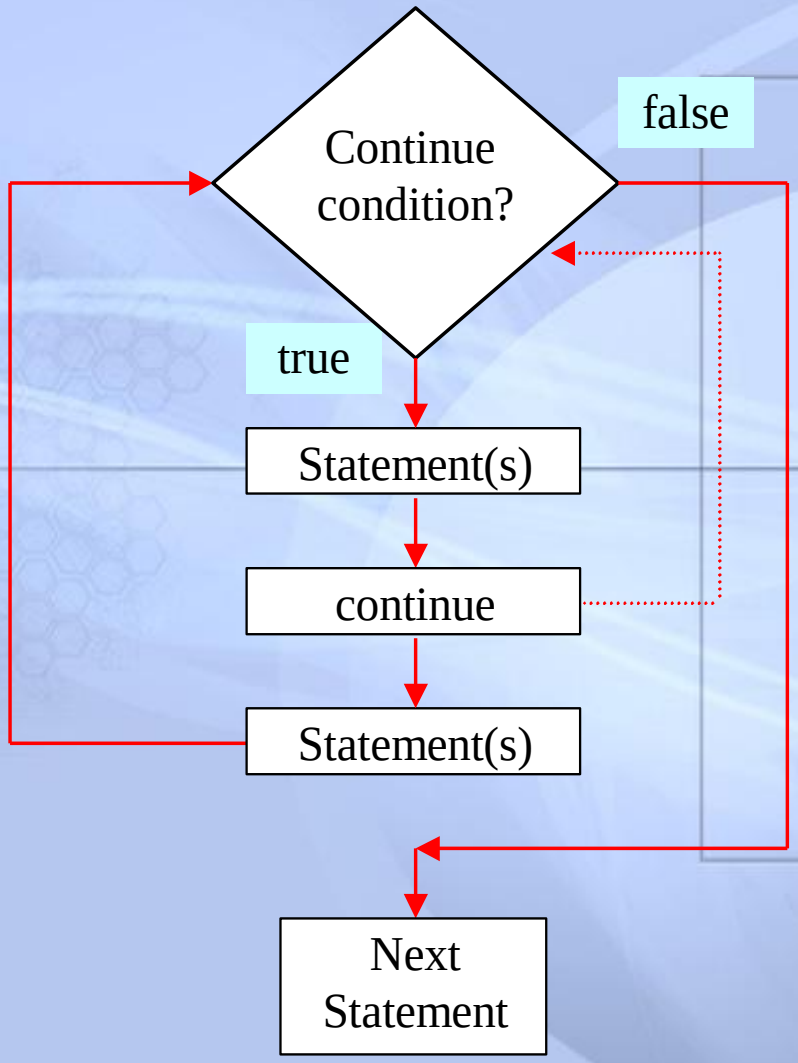


break in Loops

- Example:

```
int d = n - 1;

while (d  > 0)
{
   if (n % d == 0)
      break;
   d--;
}

if ( d > 0 )   // if found a divisor
   ...
```

The continue Keyword



Nested Loops

- A loop within a loop is called nested.

```
// Draw a 5 by 3 grid:

for (int x = 0;  x < 50;  x += 10)
{
    for (int y = 0;  y < 30;  y += 10)
    {
        g.fillRect(x, y, 8, 8);
    }
}
```

Nested Loops (cont'd)

- Braces are optional when the body of the loop(s) is one statement:

```
for (int x = 0;  x < 100;  x += 10)
  for (int y = 0;  y < 200;  y += 10)
    g.fillRect(x, y, 8, 8);
```

Inner **for** is the only statement in the outer **for**'s body

- Many programmers prefer to always use braces in loops, especially in nested loops.

Nested Loops (cont'd)

- Be careful with break:

```
int  r,  c;

for (r = 0;  r < m.length;  r++)
{
   for (c = 0;  c < m[0].length;  c++)
   {
      if (m [ r ][ c ] == 'X' )
         break;
   }
}
...
```

Breaks out of the inner loop but continues with the outer loop

Java Keywords &Reserved Words

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
package	private	protected	public	return
short	static	strictfp	super	switch
synchronized	this	throw	throws	transient
try	void	violate	while	

- **goto** and **const** are reserved: Although they have no meaning in Java.

Programming Errors

- **Syntax Errors**
  - Detected by the compiler
- **Runtime Errors**
  - Causes the program to abort
- **Logic Errors**
  - Produces incorrect result

Classes and Objects

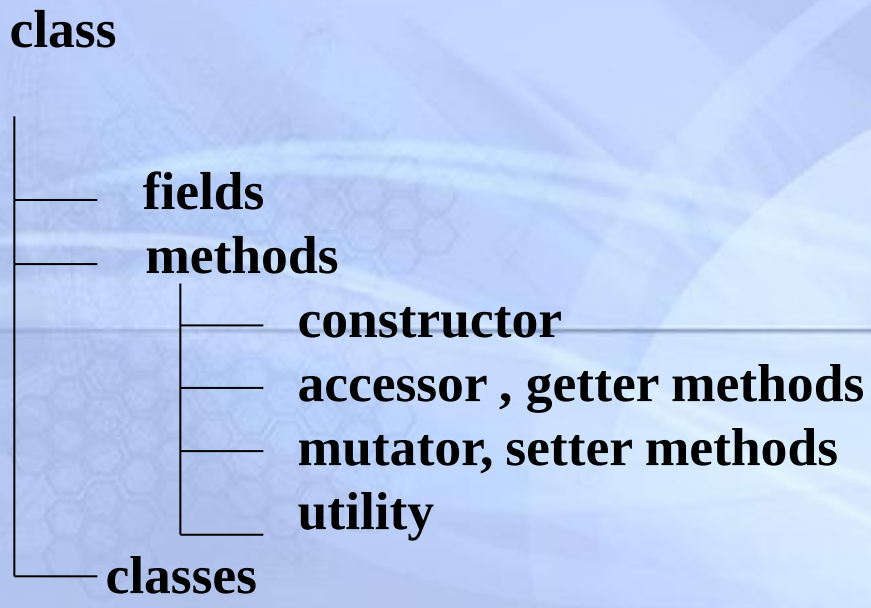
What is a class?

- A class is a template that defines the form of an object. It specifies both the data and the method that will operate on that data.
- A class is a set of plans that specify how to build an object.
- A class is a logical abstraction. It is a blueprint that resides in the Java class file.

- It has **fields**: variables with certain data types. Some data are associated with each individual object, called **instance variables**. Other data are associated with each class, called **class variables**, which are identified by keyword “static”.

- It has **methods:** the actions that act on those data. The methods also have two types. **Instance methods** are associated with each object, and **class methods** (identified as “static”) are associated with the whole class.

Simple class format



Simple class format (cont'd)

Fields – are variables that holds data.

Members – are functions that perform operations.

Constructor – is a function that creates the objects of the class. This process is called instantiation, and the resulting objects are called instances of the class. It is a duty of the constructor also to initialize the field of the objects it creates.

An accessor – method is a read only function. It may return the value of a class field without allowing it to be changed externally.

A mutator – method is a read-write function that allow external invokers to change the value of a field.

A utility – method is a private function that is used internally by other methods in its classes.

Declaring Java Class

- Basic syntax of a Java Class

```
< modifier> class < name>
{
    < attribute_declaration>*
    < constructor_declaration>*
    < method_declaration>*
}
```

Declaring Java Class (cont'd)

- Example

```
public class Book
{
    private String bookName;              //declare variable

    public Book(String inputName) {       //declare constructor
    bookName = inputName;}

    public String getBookName() {         //declare method
    return bookName;}
}
```

An example of a class

```
class Person{
   String name;
   int age;

   void birthday ( ) {
      age++;

    System.out.println (name + " is now  " + age);
   }
}
```

What is an object?

- An object is an instantiation of a class.
- According to blueprints specified in the class, an object has data (variables) and actions (methods) that act on those data.
- One or more object of the same class can be created in computer memory according to their blueprint, class.

Components of an object

- **Instance variable** = variable within an object
- **Method** = function or procedure within an object
  - Can manipulate the object's instance variables
- **Constructor** = special method to initialize a new object instance

Declaring Objects

ClassName objectName;

Example:

Person p;

Point pp;

Creating Objects

objectName = new ClassName();

Example:

p = new Person();

pp = new Point();

Declaring/Creating Objects in a Single Step

ClassName objectName = new ClassName();

Example:

Person p = new Person();

Point pp = new Point();

Accessing Objects

- Referencing the object's data:

objectName.data

p.age

pp.x

- Referencing the object's method:

objectName.method

p.birthday();

pp.setX();

Static vs. Instance Variables

- ***Variables*** or ***methods*** can be static
- Static variables are called ***class variables***
- Each class has one variable
  - No separate class variable per instance

Static vs. Instance Methods

- **Instance Methods**
  - Invoked on instances of objects
  - E.g., pp.get Y(); //invoke the getY method of ppobject
- **Static methods**
  - Invoked by preceding method name with name of class

Static Method Example

```
class X
    static void print() {
                System.out.println("This is a static method");
}}

public class Static2{
    public static void main(String args[])
    {
                X.print();
     }
}
```

Declaring Variables

- Basic Syntax of a variable

  <modifier> <type> <name> = <default_value>;

  The part in red is optional

Declaring Variables(cont.)

- Example

```
public class Book
{
        private String bookName = “No Name”;     //declare variable
        public int bookId;    //Default value
        private double  f= 0.0;
}
```

Declaring Methods

- Basic Syntax of a method

< modifier> <return_type> <name> (<parameter>*)
{
    < statement>*
}

the format of < parameter> is
< parameter_type> <parameter_name>

Declaring Methods(Cont.)

- Basic Syntax of a method

```
public class Book
{
    private String bookName;    //declare variable
    public void setBookName (String newName)
    {bookName = newName;}
    public String getBookName()  //declare method
    {return bookName;}
}
```

Access Object Members

- The ‘dot’ notation: <object>.<member>
- This is used to access object members including variables and methods
- Examples

```
javaBook.setBookName(“New Book Name”);
javaBook.bookId = 123;
```

Declaring Constructor

- Basic syntax of a constructor:

```
[< modifier>] < class_name> (<parameter>*)
{
    < statement>*
}
```

- A constructor is a set of instructions designed to initialize an instance.
- The name of the constructor must always be the same as the class name.
- Constructors are not methods. They do not have return values and are not inherited.

Declaring Constructor(cont'd)

- Example

```
public class Book
{
    private String bookName;    //declare variable
    public Book(String inputName) //declare constructor
    {bookName = inputName;}
}
```

Default Constructors

- There is always at least one constructor in every class
- If the programmer does not supply any constructors, the default constructor will be present automatically
  - The default constructor takes no arguments
  - The default constructor takes no body
- Enable you to create object instances with new XXX() with having to write a constructor

Notes: If you add a constructor declaration with arguments to a class that previously had no explicit constructors, you lose the default constructor.

A Complete Example

```
public class Book
{
    private String bookName;    //declare variable

    public Book(String inputName) //declare constructor
    {bookName = inputName;}

    public String getBookName()  //declare methods
    {return bookName;}

    public void setBookName(String inputName)
    {bookName = inputName;}
}
```

A Complete Example (cont'd)

```
public class CreateBooks  {
    public static void main(String[] args)        {

        //Book newBook1 = new Book();

        Book newBook2 = new Book("C++ Book");

        String bookName = newBook2.getBookName();
        System.out.println("The current book name is "+bookName );

        newBook2.setBookName("Java Book");
        bookName = newBook2.getBookName();
        System.out.println("The new book name is "+ bookName );
    }
}
```

Inheritance

Inheritance

- In object-oriented programming, classes are related to one another hierarchically.
- A more general class is known as ***superclass*** of a class which is more specific and is known as ***subclass***.
- A superclass has properties that are common to classes. And a subclass has the uncommon properties which distinguish a subclass from the other subclass.
- The hierarchical relationship of superclasses and subclasses are represented in Java using the *extends* keyword.

- Java does not support multiple inheritance
  - i.e., a subclass cannot be derived from multiple superclasses
  - A class can implement interfaces, however

Inheritance Hierarchy

- A class may have several subclasses and each subclass may have subclasses of its own.
- The collection of all subclasses descended from a common ancestor is called an *inheritance hierarchy.*
- The classes that appear below a given class in the inheritance hierarchy are its *descendaents.*
- The classes that appear above a given class in the inheritance hierarchy are its *ancestors.*

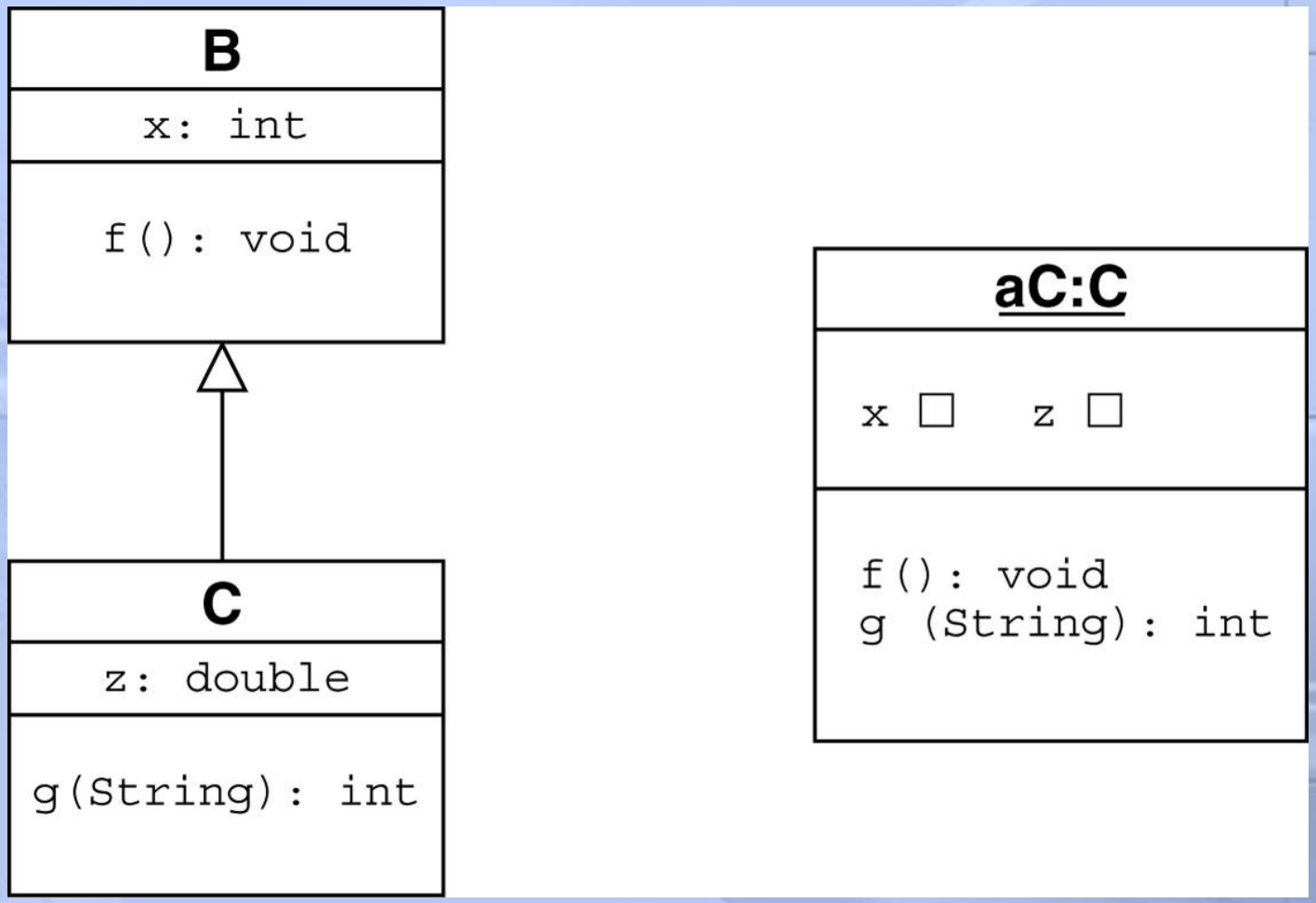
Types of Inheritance

- There are **two types of inheritance** in Java

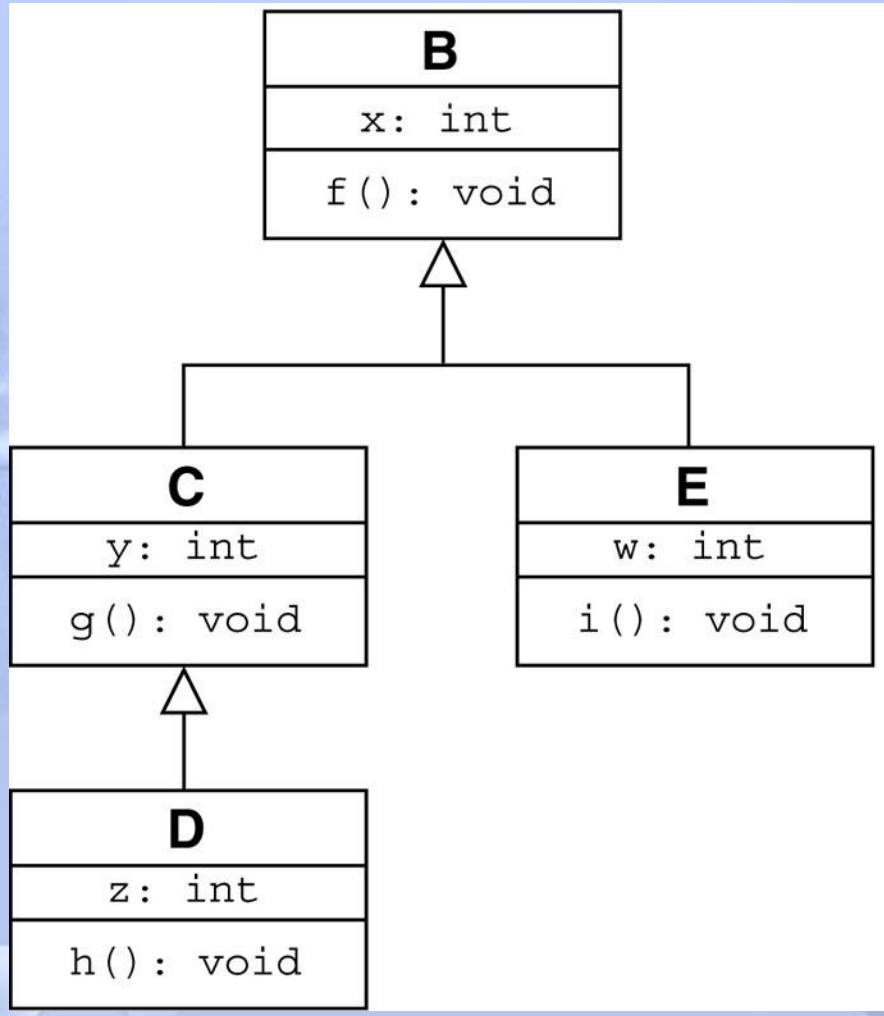
1) Single Level Inheritance

2) Multi Level Inheritance

Single Level Inheritance



Multi Level Inheritance



aC:C

x ☐ y ☐

f(): void
g(): void

aD:D

x ☐ y ☐ z ☐

f(): void
g(): void
h(): void

Constructors

- The general rule is that when a subclass is created Java will call the superclass constructor first and then call the subclass constructors in the order determined by the inheritance hierarchy.

- If a superclass does not have a default constructor with no arguments, the subclass must explicitly call the superclass constructor with the appropriate arguments

Using super( ) Call Constructor

- The call to super must be the first statement in the subclass constructor
- Example:

```
class C extends B {
  …
  public C ( … ) {
    super( B's constructor arguments );
    …
  }
  …
```

Example: Book.java

```
class Book {
  protected int pages = 1500;

  public void pageMessage() {
    System.out.println("Number of pages: " +pages);
  }
}
```

Example: Dictionary.java

```
class Dictionary extends Book {

  private int definitions = 52500;

  public void definitionMessage() {

    System.out.println("Number of definitions: " + definitions);
    System.out.println("Definitions per page: " +
                                          definitions/pages);

  }

}
```

Example: Words.java

```
class Words {
  public static void main (String[] args) {

        Dictionary webster = new Dictionary();
        webster.pageMessage();
        webster.definitionMessage();
  }
}
```

Output:

C:\Examples>java Words
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35

Extending Classes

- Protected visibility
- Super constructor
- Overriding
- Super reference
- Single vs. multiple inheritance
  - A class may have only one superclass

Example: Book2.java

```
class Book2 {
  protected int pages;

  public Book2(int pages) {
    this.pages = pages;
  }

  public void pageMessage() {
    System.out.println("Number of pages: " +pages);
  }
}
```

Example: Dictionary2.java

```
class Dictionary2 extends Book2 {
  private int definitions;

  public Dictionary2(int pages, int definitions) {
    super (pages);
    this.definitions = definitions;
  }

  public void definitionMessage () {
    System.out.println("Number of definitions: " +definitions);
    System.out.println("Definitions per page: " + definitions/pages);
  }
}
```

Example: Book3.java

```
class Book3 {
  protected String title;
  protected int pages;

  public Book3(String title, int pages) {
    this.title = title;
    this.pages = pages;
  }

  public void info() {
    System.out.println("Title: " + title);
    System.out.println("Number of pages: " + pages);
  }
}
```

Example: Dictionary3b.java

```
class Dictionary3b extends Book3 {
  private int definitions;

  public Dictionary3b(String title, int pages, int definitions) {
    super (title, pages);
    this.definitions = definitions;
   }

  public void info() {
    super.info();
    System.out.println("Number of definitions: " + definitions);
    System.out.println("Definitions per page: " + definitions/pages);
   }
}
```

Example: Books.java

```
class Books {
  public static void main (String[] args) {
    Book3 java = new Book3("Introduction to Java", 350);
    java.info();

    System.out.println();
    Dictionary3a webster1 = new Dictionary3a("Webster English
                                              Dictionary", 1500, 52500);
    webster1.info();

 System.out.println();
    Dictionary3b webster2 = new Dictionary3b("Webster English
                                              Dictionary", 1500, 52500);
    webster2.info();
  }
}
```

Example: Books.java

Output:

```
C:\Examples>java Books
Title: Introduction to Java
Number of pages: 350

Dictionary: Webster English Dictionary
Number of definitions: 52500
Definitions per page: 35

Title: Webster English Dictionary
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35
```

Thank You!!!