Introduction

Abstraction

- An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior.
- A powerful way to manage abstraction is through the use of hierarchical classifications.
- This allows you to layer the semantics of complex systems, breaking them into more manageable pieces.
- From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on
- Object-oriented concepts form the heart of Java just as they form the basis for human understanding. It is important that you understand how these concepts translate into programs.

The Three OOP Principles

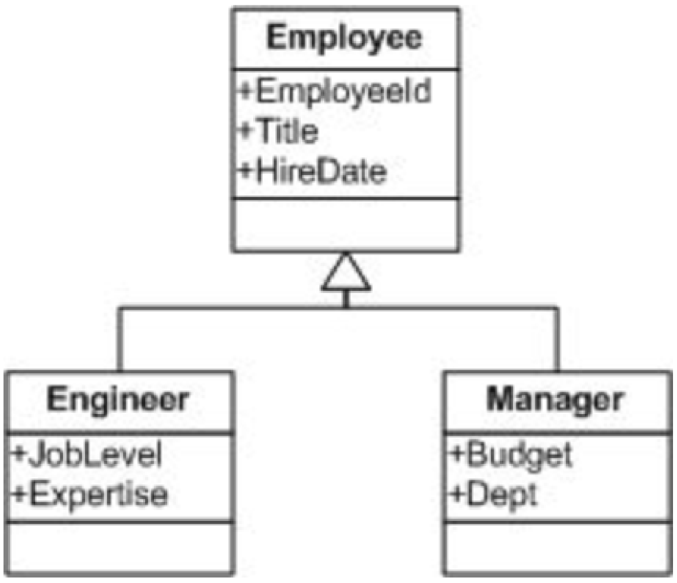
Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper
- Access to the code and data inside the wrapper is tightly controlled through a well-defined interface

- To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about your engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever. You, as the user, have only one method of affecting this complex encapsulation: by moving the gear-shift lever.
- Thus, the gear-shift lever is a well-defined (indeed, unique) interface to the transmission
- Further, what occurs inside the transmission does not affect objects outside the transmission. For example, shifting gears does not turn on the headlights!
- In Java, the basis of encapsulation is the class (covered later).
- A class defines the structure and behavior (data and code) that will be shared by a set of objects
- Each object of a given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality.

Inheritance

- Inheritance is the process by which one object acquires the properties of another object.
- This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications.
- For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal.
- Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent

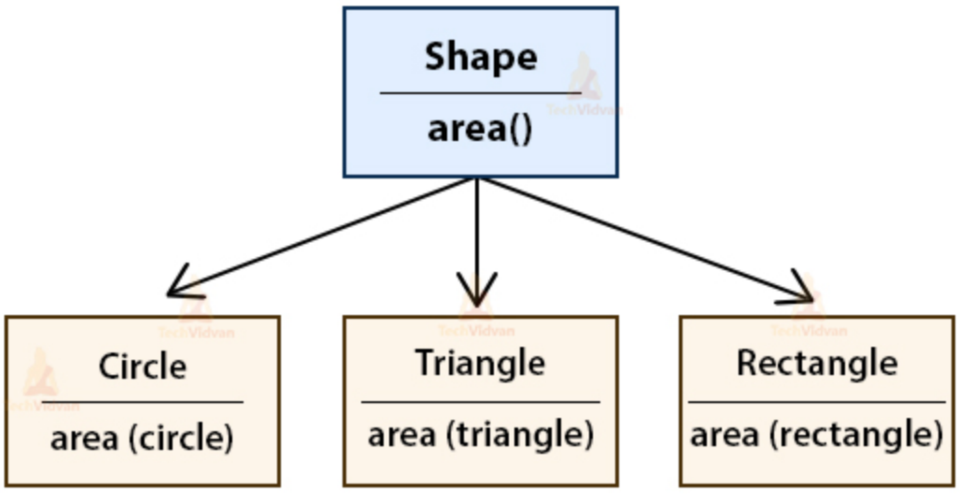


Polymorphism

- Polymorphism (from Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions.
- Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stacks. One stack is used for integer values, one for floating- point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs.
- In a non–object-oriented language, you would be required to create three different sets of stack routines, with each set using different names.
- However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

- More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a generic interface to a group of related activities.

  It is the compiler’s job to select the specific action (that is, method) as it applies to each situation. You, the programmer, do not need to make this selection manually.



A First Simple Program

```
/* This is a simple Java program. Call this file "Example.java".
*/ class Example { // Your program begins with a call to main()
public static void main(String args[]) { System.out.println("This
is a simple Java program."); } }
```

Java

- The first thing that you must learn about Java is that the name you give to a source file is very important. For this example, the name of the source file should be **Example.java**.
- As you can see by looking at the program, the name of the class defined by the program is also Example. This is not a coincidence. In Java, all code must reside inside a class. By convention, the name of the main class should match the name of the file that holds the program. You should also make sure that the capitalization of the filename matches the class name. The reason for this is that Java is case-sensitive. At this point, the convention that filenames correspond to class names may seem arbitrary. However, this convention makes it easier to maintain and organize your programs.
- To compile the Example program, execute the compiler, javac, specifying the name of the source file on the command line, as shown here:

```
javac Example.java
```

Java

- The javac compiler creates a file called Example.class that contains the bytecode version of the program.
- To actually run the program, you must use the Java application launcher called java. To do so, pass the class name Example as a command-line argument, as shown here:

```
java Example
```

Java

- When you execute java as just shown, you are actually specifying the name of the class that you want to execute. It will automatically search for a file by that name that has the .class extension. If it finds the file, it will execute the code contained in the specified class.

A Closer Look at the First Sample Program

- Keyword **class** used to declare classes
- **Example** is the name of the class

- Now pay attention to this line

```java
public static void main(String args[ ])
```

Java

- This is the line at which the program will begin executing.
- All Java applications begin execution by calling **main( )**
- **public** keyword is an access modifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared. (The opposite of **public** is **private**, which prevents a member from being used by code defined outside of its class.)
- In this case, **main( )** must be declared as public, since it must be called by code outside of its class when the program is started.
- The keyword static allows **main( )** to be called without having to instantiate a particular instance of the class
- The keyword void simply tells the compiler that **main( )** does not return a value.
- Keep in mind that Java is case-sensitive. Thus, Main is different from main.
- In main( ), there is only one parameter, albeit a complicated one. String args declares a parameter named args, which is an array of instances of the class String. (Arrays are collections of similar objects.)
- One other point: main( ) is simply a starting place for your program. A complex program will have dozens of classes, only one of which will need to have a main( ) method to get things started.
- All statements in Java end with a semicolon

A Second Short Program

- The program and it's output is as follows

```java
/* Here is another short example. Call this file "Example2.java".
*/ class Example2 { public static void main(String args []) { int
num; // this declares a variable called num num = 100; // this
assigns num the value 100 System.out.println("This is num: " +
num); num = num * 2; System.out.print("The value of num * 2 is
"); System.out.println(num); } }
```

Java

```
This is num: 100 The value of num * 2 is 200
```

Plain Text

- Java (like most other languages) requires that variables be declared before they are used.
- String concatenation done while printing output.
- Actually, num is first converted from an integer into its string equivalent and then concatenated with the string that precedes it.
- This approach can be generalized. Using the + operator, you can join together as many items as you want within a single println( ) statement
- The print( ) method is just like println( ), except that it does not output a newline character after each call

Two Control Statements

The if Statement

```
if(condition) statement;
```

Plain Text

- Here, *condition* is a Boolean expression. If *condition* is true, then the statement is executed.If *condition* is false, then the statement is bypassed.

```
if(num < 100) System.out.println("num is less than 100");
```

Java

```
/* Demonstrate the if. Call this file "IfSample.java". */ class
IfSample { public static void main(String args[]) { int x, y; x =
10; y = 20; if (x < y) System.out.println("x is less than y"); x
= x * 2; if (x == y) System.out.println("x now equal to y"); x =
x * 2; if (x > y) System.out.println("x now greater than y"); //
this won't display anything if (x == y) System.out.println("you
won't see this"); } }
```

Java

```
x is less than y x now equal to y x now greater than y
```

Plain Text

The for Loop

```
for(initialization; condition; iteration) statement;
```

Plain Text

- In its most common form, the initialization portion of the loop sets a loop control variable to an initial value. The condition is a Boolean expression that tests the loop control variable. If the outcome of that test is true, the for loop continues to iterate. If it is false, the loop terminates. The iteration expression determines how the loop control variable is changed each time the loop iterates

```
/* Demonstrate the for loop. Call this file "ForTest.java". */
class ForTest { public static void main(String args[]) { int x;
for (x = 0; x < 10; x = x + 1) System.out.println("This is x: " +
x); } }
```

Java

```
This is x: 0 This is x: 1 This is x: 2 This is x: 3 This is x: 4
This is x: 5 This is x: 6 This is x: 7 This is x: 8 This is x: 9
```

Plain Text

- x = x+1 can be replaced with x++ (The increment operator is ++)
- The increment operator increases its operand by one
- Java also provides a decrement operator, which is specified as – –. This operator decreases its operand by one

Using Blocks of Code

- Java allows two or more statements to be grouped into blocks of code, also called code blocks. This is done by enclosing the statements between opening and closing curly braces.
- Once a block of code has been created, it becomes a logical unit that can be used any place that a single statement can.

```java
if(x < y) { // begin a block x = y; y = 0; } // end of block
```

Java

```java
/* Demonstrate a block of code. Call this file "BlockTest.java"
*/ class BlockTest { public static void main(String args[]) { int
x, y; y = 20; // the target of this loop is a block for (x = 0; x
< 10; x++) { System.out.println("This is x: " + x);
System.out.println("This is y: " + y); y = y - 2; } } }
```

Java

```
This is x: 0 This is y: 20 This is x: 1 This is y: 18 This is x:
2 This is y: 16 This is x: 3 This is y: 14 This is x: 4 This is
y: 12 This is x: 5 This is y: 10 This is x: 6 This is y: 8 This
is x: 7 This is y: 6 This is x: 8 This is y: 4 This is x: 9 This
is y: 2
```

Plain Text

Input from keyboard

Scanner

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class.

Reading a single integer number from keyboard

```
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt(); System.out.println(n); } }
```
Java

Input

```
4
```
Java

Output

```
4
```
Java

Reading a single double value from keyboard

```
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
double n = scanner.nextDouble(); System.out.println(n); } }
```
Java

Input

```
121.232
```
Java

Output

```
121.32
```
Java

Reading 2 integers from keyboard

Sample Input

```
10 5
```
Java

or

```
10 5
```

Java

Output

```
10 5
```

Java

```java
public class InputReader { public static void main(String[] args) {
Scanner scanner = new Scanner(System.in); int n = scanner.nextInt();
int d = scanner.nextInt(); System.out.println(n);
System.out.println(d); } }
```

Java

Reading a string input from keyboard

```java
public class InputReader { public static void main(String[] args) {
Scanner scanner = new Scanner(System.in); String name =
scanner.nextLine(); System.out.println(name); } }
```

Java

Input

```
Masai School is Awesome
```

Java

Output

```
Masai School is Awesome
```

Java

Reading 2 integer numbers and a string

Input

```
2 4 Lloyd
```

Java

```
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt(); int d = scanner.nextInt();
scanner.nextLine(); String name = scanner.nextLine();
System.out.println(n); System.out.println(d);
System.out.println(name); } }
```
Java

Output

```
2 4 Lloyd
```
Java

Reading an 1D Array through Keyboard

Input

```
Take a value n from keyboard which represents the size of the array
and create a array of size n and store all the elements in the array
and display it 5 10 20 30 40 50
```
Java

Output

```
10 20 30 40 50
```
Java

```
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt(); int[] a = new int[n]; for (int i = 0; i <
a.length; i++) { a[i] = scanner.nextInt(); } for (int i = 0; i <
a.length; i++) { System.out.print(a[i] + " "); } } }
```
Java

Reading an 2D array

```java
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
System.out.println("Enter the row size"); int rowSize =
scanner.nextInt(); System.out.println("Enter the column size"); int
columnSize = scanner.nextInt(); int[][] a = new int[rowSize]
[columnSize]; System.out.println("Enter the array elements"); for
(int i = 0; i < a.length; i++) { for (int j = 0; j < a[i].length;
j++) { a[i][j] = scanner.nextInt(); } } System.out.println("The array
Elements are"); for (int i = 0; i < a.length; i++) { for (int j = 0;
j < a[i].length; j++) { System.out.print(a[i][j] + " "); }
System.out.println(); } } }
```
Java

Output

```
Enter the row size 2 Enter the column size 3 Enter the array elements
10 20 30 40 50 60 The array element are 10 20 30 40 50 60
```
Java

Reading a integer as a string and parsing it to int

```java
import java.util.Scanner; public class InputReader { public static
void main(String[] args) { Scanner scanner = new Scanner(System.in);
System.out.println("Enter a number "); String number =
scanner.nextLine(); int n = Integer.parseInt(number);
System.out.println("The number is " + n); } }
```
Java

Output

```
Enter a number 12 The number is 12
```
Java

Note

Please make sure that whenever you parse a string to integer, you have entered a number or else exception will be thrown.

Reading an array as a string and parsing it to array

```
public class InputReader { public static void main(String[] args) {
Scanner scanner = new Scanner(System.in); System.out.println("Enter
the array elements"); String name = scanner.nextLine(); String[] arr
= name.split(" "); int[] a = new int[arr.length]; for (int i = 0; i <
arr.length; i++) { a[i] = Integer.parseInt(arr[i]); }
System.out.println("Entered array elements are"); for (int i =0 ;
i<a.length ; i++){ System.out.print(a[i] + " "); } } }
```

Java

Output

```
Enter the array elements 10 20 30 40 50 Entered array elements are 10
20 30 40 50
```

Java