Chapter-5 :

Objects in a program can represent either objects in the real world or abstractions;

A class is the definition of a kind of object;

An instance of a class is an object;

A class specifies the attributes, or data, that objects of the class have.

The class also specifies what actions the objects can take and how they accomplish those actions;

Class files and separate compilation:

The name of the file should begin with the name of the class and end in .java;

You can compile a java class before you have a program in which to use it . The compiled bytecode for the class will be stored in a file of the same name , but ending in .class rather than .java;

Instance variables and methods are members of a class;

New:

New is a unary operator that we use to create objects of a class.

A program is simply a class that has a method named main.

Methods:

Java has two kinds of methods:

Methods that return a single item;

Methods that perform some action other than returning an item;

Methods nextInt is an example of a method that returns a single value, one of type int; The method println is an example of a method that performs some action other than returning a single value. Method that perform some action other than returning a value are called void methods;

You invoke a method by writing the name of the receiving object followed by a dot, the name of the method, finally a set of parentheses that can contain arguments providing information for the method;

Calling a method that returns a quantity:

If a method returns a single quantity, you can invoke it anywhere that you can use a value of the type returned by the method .

If a method performs some action other than returning a single quantity, you write its invocation followed by a semicolon. The resulting Java statement performs the action defined by the method.

Defining void methods

A method definition has a heading and a body;

Local variables are those declared within a method’s body;

When a void method is invoked, it is as if the method invocation were replaced by the body of the method definition and its statements were then executed.

The heading for a method that returns a value is almost the same as the heading for a void method; The only difference is that a method that returns a value uses a type name instead of the keyword void. The heading begins with the keyword public , followed by the name of the method and a pair of parentheses;

The body of a method definition that returns a value is just like the body of a void method definition, except that it must contain at least one return statement:

Return expression;

A return statement says that the value returned by the method is the value of Expression. Expression can be any expression that produces a value of the type specified in the heading of the method definition.

When a return statement executes, not only does it provide the value returned by the method but it also ends the method’s execution. If more statements follow the return statement , they are not executed;

A method that returns a value may perform some other action as well, such as reading a value from the keyboard, but it definitely must return a value;

If(age <= 2)

{

Return age \*11;

}

Else{

Return 22+ ((age -2)\*5);

}

In other programming languages., Methods that return a value are called functions;

Since a void method returns no value , it typically does not have any return statement. However, you can write a return statement within a void method without an accomopanying expression, as follows: return ;

Every method belongs to some class and is available to objects created from that class. The definition of a method is given in the definition of the class to which it belongs. Each method either returns a single quantity or does not return any value.

Public Return\_Type Method\_Name(Parameters)

{

Statements}

Return Expression;

the keyword this:

Inside the definition of a method of the same class , you can simply use the instance variable name without any object name or dot.

Every instance variable , including name , is an instance variable of some object .

Although this is frequently omitted, but understood to be there, you can include it if you want; this.name = keyboard.nextLine();

The keyword this stands for the receiving object;

Within a method definition , you can use the keyword this as a name for the object receiving the method call;

Local variables:

A variable declared within a method is called a local variable.

Java has no global variables;

Local variables:

A variable declared within a method definition is called a local variable. One method’s local variables have no meaning within another method. If two methods each have a local variable with the same name ,they are considered two different variables.

Instance variables, whose meaning is confined to an object of a class;

Local variables, whose meaning is confined to a method definition.

A block is a compound statement that declares a local variable;

If you declare a variable within a block that is within a compound statement—the variable is local to the block;

Formal parameters:

Public int predictPopulation( int years)

The world years is a formal parameter. It is a stand-in for a value that will be plugged in when the method is called. The item that is plugged in is called an argument ; Arguments are called actual parameters;

Java passes arguments to a method using call-by-value;

Parameter names are local to the method;

Formal parameters are really local variables, and so their meanings are confined to their respective method definitions;

Automatic type conversion - --

Parameters of a class type behave differently from parameters of a primitive type.

Arguments must match parameters in number , order, and type;

Within a method definition, formal parameters are given within the parentheses after the method name. In a method invocation, arguments are given within the parentheses after the method name. Arguments must match the formal parameters in the method heading with respect to their number, their order, and their data types;

Java will perform an automatic type conversion when the types do not match exactly;

5.2 Information hiding and encapsulation

A programmer who is using a method that you have defined does not need to know the details of the code in the body of the method definition to be able to use the method ;

Designing a method so that it can be used without any need to understand the fine detail of the code is called information hiding;

Comments:

A programmer who uses a method should need only know what the method does, not how it does it ;

Precondition and Postcondition comments:

A method’s precondition comment states the conditions that must be true before the method is invoked. The method should not be used, and can not be expected to perform correctly, unless the precondition is satisfied.

The postcondition comment describes all the effects produced by a method invocation. The postcondition tells what will be true afrer the method is executed in a situation in which the precondition holds.

For a method that returns a value , the postcondition will include a description of the value returned by the method;

You should always think in terms of preconditions and postconditions when writing method comments;

The modifier public, when applied to a class , method ,or instance variable, means that any other class can directly use or access the class, method, or instance variable by name ;

Typically, all instance variables should be private .

The keywords public and private are examples of access modifiers;

When an instance variable is private , its name is not accessible outside of the class definition.

Even so, you can still use its name in any way you wish within any method inside the class definition. However the invocation of the method is valid; ( if private , not access from outside, but the invocation of the method is valid) ;

Thus, making an instance variable private does not mean that you can not change it ; It means only that you can not use the instance variable’s name to refer directly to the variable anywhere outside of the class definition;

Methods can also be private. If a method is marked private, it can not be invoked outside of the class definition. However, a private method can still be invoked within the definition of any other method in that same class.

Conclusion:

If an instance variable is private, its name can not be used to access it outside of the class definition; However ,it can be used within the definitions of methods in its class. If a instance variable is public , there are no restrictions on where you can use its name;

If a method definition is private, the method cannot be invoked outside of the class definition. However, it can be invoked within the definitions of methods in its class. If the method is public , you can invoke it anywhere without restriction;

Normally, all instance variables are private and most methods are public ;

Public instance variables can lead to the corruption of an object’s data;

Private instance variables enable the class to restrict how they are accessed or changed;

Accessor methods and mutator methods:

An accessor method or get method or getter , is simply a method that allows you to look at data contained in an instance variable. getName;

Accessor methods allow you to look at the data in a private instance variable. Other methods known as mutator methods or set methods, or setters, allow you to change the data stored in private instance variables. setSpecies ;

It is traditional to begin the names of accessor methods with get and mutator methods with set;

A method body can contain an invocation of another method. If the called method is in the same class , it is typically invoked without writing any receiving object. This is true regardless of whether the called method is public or private;

If a method is labeled private, it can be called only from within the definitions of other methods of the same class;

Encapsulation:

Encapsulation groups instance variables and methods into a class;

Encapsulation hides implementation details;

Only the necessary controls are visible and its details are hidden;

The class interface tells programmers all they need to know to user the class in their programs. The class interface consists of the headings for the public methods and public named constants of the class , along with comments that tell a programmer how to use these public methods and constants;

Guidelines for defining a well-encapsulated class:

1.Place a comment before the class definition that describes how the programmer should think about the class data and methods;

2.Declare all the instance variables in the class as private;

3.Provide public accessor methods to retrieve the data in an object. Also provide public methods for any other basic needs that a programmer will have for manipulating the data in the class.

4.Place a comment before each public method heading that fully specifies how to use the method;

5.make any helping methods private;

6.write comments within the class definition to describe implementation details;

API application programming interface: The API for a class is essentially the same thing as the class interface for the class. You will often see the term API when reading the documentation for class libraries;

Encapsulation means that data and actions are combined into a single item, a class object that hides the details of the implementation.

Variables of a class type behave very differently from variables of a primitive type; Variables of a class type are names for objects of their class, but the objects are not the values of the variables in the same way;

Variables of a class type name objects in a way that is different from how variables of a primitive type or a class type store their values; Every variable whether of a primitive type or a class type , is implemented as a memory location .

Class types are reference types ;

A variable of a class type contains the memory address of the object named by the variable;

The object itself is not stored in the variable , but rather in some other location in memory; The address of this other memory locations is called a reference to the object; For this reason, class types are often called references types. A reference type is a type whose variables hold references—that is memory addresses--- as opposed to actual values of objects;

The reason why variables of a primitive type and variables of a class type name behave in different ways. Every value of a given primitive , always requires the same amount of memory. However, an object ,might be any size. Making it difficult for the system to set aside a fixed amount of space for variables that name objects;

Variables of a class type store memory addresses for objects, not the objects themselves;

When assigning ; copy the memory address from variables of objects to variables of objects ;

A memory address is a number;

Conclusion:

A variable of a class type contains the memory address of the object’s location in the computer’s memory; This scheme allows a variable of a class type to be used as a name for an object of that class;

A variable of a class type holds the address in memory of an object of that class. This memory address is often called a reference to the object in memory; Thus class types are reference types. Variables of a reference type hold references that is memory addresses as opposed to actual values of objects; All class types are reference types , but not all reference types are class types;

Boolean-Valued Methods:

Methods can return a value of type Boolean.

Parameters of a Class Type

A method’s parameters of a class type are treated differently than its parameters of a primitive type;

Formal parameters are given in parentheses after the method name at the beginning of a method definition. A formal parameter of a class type is a local variable that holds the memory address of an object of that class type .

A method can not change the value of an argument of a primitive type that is passed to it . In addition, a method can not replace an object passed to it as an argument with another object. On the other hand, a method can change the values of the instance variables of an argument of a class type .