Chapter-6

6.1 Constructor

When you create an object of a class using the new operator, you invoke a special kind of method called a constructor;

Constructors create and initialize new objects;

A constructor is a special method that is called when you use the new operator to create a new object.

Species earthSpecies = new Species();

The first part of this statement Species earthSpecies, declares the variable earthSpecies to be a name for an object of the class species. The second part , newSpecies(), creates and initializes a new object, whose address is then assigned to earthSpecies. Species() is a call to the constructor that Java provided for the class; The parentheses are empty because this particular constructor takes no arguments

A constructor can perform any action you write into its definition, but a constructor is meant to perform initializing actions; Constructors serve essentially the same purpose as set methods;

But unlike set methods, constructors create an object as well as initializing it . Like set methods, constructors can have parameters;

Instance variables that you do not initialize are given default initial values by Java;

One property of constructors that may seem strange at first is that each constructor has the same name as its class;

So if the class is named Species, its constructors are named Species.

Constructors often have multiple definitions, each with different numbers or types of parameters and sometimes they parallel a class’s set methods;

The headings of these constructors do not have the word void. When you define a constructor , you do not specify any return type, not even void;

Public pet(String initialName, int initialAge) do not have void;

When you define a constructor , you do not specify any return type, not even void.

Constructors are called only when you create an object. To change the state of an existing object , you need one or more set methods;

However , if you define at least one constructor in a class, no constructors are created for you automatically;

When you create a new object by using the operator new , you must always include a call to a constructor;

As with any method invocation, you list any arguments in parentheses after the constructor’s name -which remember is the same as the class name;

E.g: Pet fish = new Pet( “Wanda” , 2, 0.25);

Pet myPet = new Pet(); creates a new object of the class Pet by calling the default constructor, that is the constructor without parameters;

You can not use an existing object to call a constructor, so

myPet.Pet(“Fang” , 1, 150.0 ) ; is invalid

but if you need some other way to change the values of the instance variables of an object, once it has been created; That way involves calling one or more set methods;

myPet.setPet(“Fang” ,1, 150.0);

A constructor is a method that is called when an object of the class is created by using the new operator.

Constructors must have the same name as the class to which they belong;

A constructor can call methods within its class.

You have to be careful when constructors call public methods of their class ;

The problem has to do with inheritance;

Another solution to avoid repeating code is to make the method that a constructor calls private;

When defining a constructor for a class, you can use this as a name for another constructor in the same class. Any call to this must be the first action taken by the constructor;

6.2: Static variables and static methods:

A class can have variables that are static as well as methods that are static.

Static variables:

Public static final double FEET\_PER\_YARD = 3;

The particular static variable can not change in value; it is a constant because its definition includes the keyword final ;

The following static variable can change in value:

Private static int numberOfInvocations;

A static variable can be public or private;

Static variables like instance variables that are not constants should normally be private and should be accessed or changed only via accessor and mutator methods;

Static variables are also called class variables;

The declaration of a static variable contains the keyword static. A static variable is shared by all the objects of its class;

Static methods:

When you define a static method, the method is still a member of a class, since you define it in a class, but the method can be invoked without using any object. You normally invoke a static method by using the class name instead of an object name;

You can create a collection of static methods to perform computations that are somehow related and group them within a single class;

A static method can not reference an instance variable;

A static method can call only a static method ; main is a static method;

A non-static method can reference any variable or call any method within its class;

You cannot invoke a non-static method within a static method unless you have an object to use in the call of the non-static method.

Summary about the static method:

The static method can be invoked using the class name in place of an object name . Since it does not need an object in its invocation, a static method cannot reference an instance variable of the class . It also cannot invoke a non-static method of the class , unless it has an object of the class and uses this object in the invocation.

Helping methods for a main method;

Math class :

Math.PI;

Math.E;

Floor: rounding down; ceil : rounding up ;

The method random returns a random double that is >= 0 and < 1;

Math.random();

Wrapper Classes:

If a method needs an argument of a class type, but we have a value of a primitive type , we need to convert the primitive value such as the int value 42 –to an equivalent value of some class type that corresponds to the primitive type int; To make this conversion, Java provides a wrapper class for each of the primitive types; Such classes define methods that can act on values of a primitive type;

Integer is a wrapper class for the primitive type int;

Integer n = new Integer(42);

And the method is intValue()

Int i = n.intValue();

Static methods in the wrapper classes can be used to convert a string to the corresponding number of type int, double, long or float;

The static method parse Double of the wrapper class Double will convert a string to a value of type double.

Double.parseDouble(“199.98”)

Returns the double value 199.98;

Each of the numeric wrapper classes also has a static method called toString that will convert in the other direction—that is , it converts from a numeric vaue to a string representation of the numeric value;

Integer.toString(42);

Character is the wrapper class for the primitive type char.

Character c1 = new Character(‘a’);

We will not use the wrapper class Boolean; Because it’s more complicated;

Every primitive type has a wrapper class. Wrapper classes allow you to have a class object that corresponds to a value of a primitive type . Wrapper classes also contain a number of useful predefined constants and methods. Thus, each wrapper class has two related but distinct uses;

Key:

The wrapper classes Boolean , Byte ,Character, Double ,Float, Integer, Long and Short have no default constructors.

Thus Integer n = new Integer(); is invalid;

When you create a new object of one of these classes, you must provide an initializing value as an argument to the constructor, as in the following example:

Character myMark = new Character(‘Z’);

Wrapper classes do not have set methods, so you cannot change the value of existing objects of these classes. The designers, therefore , decided not to allow the creation of objects that have default values, and so they did not define default constructors;

Decomposition

Programming tip:

Every time you change the definition of a class or method , you should test it ;

Addressing compiler concerns

The compiler checks to make sure you have done certain necessary things, such as initializing variables or including a return statement in the definition of a method that returns a value;

String line = null;

The constant value null is a special predefined constant that you can use to give a value to any variable of any class type ; Because null is like an address; you use == or != rather than the method equals when you test to see whether a variable contains null ;

Since null is not a object you can not use it to invoke a method. If you try, you will get an error message that says “ Null pointer Exception.”

Every method you write for a class should be tested.

**6.4 Overloading:**

Overloading a method name means giving the same name to more than one method within a class;

A method’s name and the number and types of its parameters are called the method’s signature;

The methods in a class must have different signatures. A class cannot define two or more methods with the same signature. A method’s signature does not include its return type;

Overloading can be applied to any kind of method. It can be applied to void methods, to methods that return a value, to static methods, to non-static methods, or to any combination of these; You can also overload constructors.

Public class SampleClass

{

Public static void problemMethod(double n1, int n2)

Public static void problelMethod(int n1, double n2)

}

And the invocation such as

SampleClass.problemMethod(5,10) will produce an error message;

You cannot overload a method name by giving two definitions whose headings differ only in the type of the value returned; Such as:

Public double getWeight()

Public char getWeight()