Objects and Classes

Part 6 – Object Construction

Chapter 4, Core Java, Volume I

Contents

- Object Construction
- Constructor Overloading
- Default Constructor
- Explicit Field Initialization
- Initialization Blocks
- Constructor Parameter Names
- Calling Another Constructor
- The Order of Initialization
- Constructor Test

Object Construction

- Three ways to initialize an instance field:
  - By setting a value in a constructor
  - By assigning a value in the declaration
  - By setting a value in an initialization block

- A field that isn't explicitly initialized is set to a default: `0`, `false`, or `null`.
- Caution: Accidentally uninitialized reference variables can lead to null pointer exception.

```
public Employee() { name = ""; }
. . .
LocalDate h = harry.getHireDay();
int year = h.getYear();
```

- Caution: local variables must be explicitly initialized before use

Constructor Overloading

- A class can have more than one constructor:

```
StringBuilder messages = new StringBuilder();
StringBuilder todoList = new StringBuilder("To do:\n");
```

- The constructor name is *overloaded*.

```
class Account
{
   private String name;
   private double balance;

   public Account(String n, double b)
   {
       name = n;
       balance = s;
    }
```

```
   public Account(String n)
   {
       name = n;
       balance = 0.0;
   }
   . . .
}

Account a1 = new Account(“Lee”, 100000.0);
Account a2 = new Account(“Park”);
```

Default Constructor

- If a class has no constructor, a default constructor (no-argument constructor) is provided.
  - It sets all fields to their default values.
- If a class has at least one constructor, the default constructor is not provided.
- But you can provide a no-argument constructor:

```
public Employee() {}

or

public Employee()
{
   name = “”;
   salary = 0;
   hireDay = LocalDate.now();
}
```

Explicit Field Initialization

- You can initialize fields explicitly in the declaration:
  - initialized before the constructor executes

```
class Employee
{
   private String name = "";
   . . .
}
```

- A field can be initialized with a method call:

```
class Employee
{
   private static int nextId;
   private int id = assignId();
   . . .
   private static int assignId()
   {
      int r = nextId;
      nextId++;
      return r;
   }
   . . .
}
```

Initialization Blocks

- Class declarations can contain arbitrary blocks of code.
- Executed whenever an object is constructed (before the body of constructor):

```
private static int nextId;
private int id;
// object initialization block
{
   id = nextId;
   nextId++;
}
public Employee(. . .) { . . . }
. . .
```

- Static initialization block is executed when class is loaded:

```
static
{
   Random generator = new Random();
   nextId = generator.nextInt(10000);
}
```

Constructor Parameter Names

- Some programmers like single-letter or short-length parameter names:

```
public Employee(String n, double s)
{
   name = n;
   salary = s;
}
```

- For better documentation, can use a prefix:

```
public Employee(String aName, double aSalary)
{
   name = aName;
   salary = aSalary;
}
```

- Can use this to distinguish between fields and parameters with the same names:

```
public Employee(String name, double salary)
{
   this.name = name;
   this.salary = salary;
}
```

Calling Another Constructor

- A constructor can call another constructor in the first statement.
- Use this (and not the class name) for the call:

```
public Employee(double s)
{
   this("Employee #" + nextId, s); // calls Employee(String, double)
   nextId++;
}
public Employee(String n, double s)
{ … }
```

- Allows you to factor out common construction code.
- Keyword reuse: Not related to using this for the implicit parameter.

The Order of Initialization

1. All data fields are initialized to their default values.
2. All field initializers and initialization blocks are executed in the order in which they occur in the class.
3. If the first line of the constructor calls a second constructor, then the body of the second constructor is executed.
4. The body of the constructor is executed

Constructor Test

```
public class ConstructorTest
{
   public static void main(String[] args)
   {
      // fill the staff array with three Employee objects
      Employee[] staff = new Employee[3];

      staff[0] = new Employee("Harry", 40000);
      staff[1] = new Employee(60000);
      staff[2] = new Employee();

      // print out information about all Employee objects
      for (Employee e : staff)
         System.out.println("name=" + e.getName() + ",id=" + e.getId() + ",salary="
               + e.getSalary());
   }
}
```

Constructor Test

```
class Employee
{
   private static int nextId;

   private int id;
   private String name = ""; // instance field initialization
   private double salary;

   // static initialization block
   static
   {
      Random generator = new Random();
      // set nextId to a random number between 0 and 9999
      nextId = generator.nextInt(10000);
   }

  // object initialization block
   {
      id = nextId;
      nextId++;
   }
```

```
   // three overloaded constructors
   public Employee(String n, double s)
   {
      name = n;
      salary = s;
   }
   public Employee(double s)
   {
      // calls the Employee(String, double) constructor
      this("Employee #" + nextId, s);
   }
   // ~~the default constructor~~
   public Employee()
   {
      // name initialized to ""--see above
      // salary not explicitly set--initialized to 0
      // id initialized in initialization block
   }
  // getters …
}
```