II. Типы, переменные, управляющие инструкции

3. Ссылочные типы и переменные

Ссылочные типы и объекты

Существует три вида ссылочных типов: типы классов, типы интерфейсов и типы массивов. Переменные ссылочных типов предназначены для хранения ссылок на объекты. Можно объявлять свои типы классов и интерфейсов.

Объект – зкземпляр класса или массива. Объекты можно создавать с помощью оператора new. Оператор new возвращает ссылку на созданный объект. Память для хранения объектов выделяется в специальной области памяти называемой управляемой кучей. Память объектов которые уже не будут использоваться автоматически освобождается с помощью специального процесса который называется сборка мусора.

Ссылочные переменные

```
Class variable [= new Class ([param-list]) ];
```

Значением переменной ссылочного типа является ссылка на объект. Переменная типа класса может использоваться для хранения ссылки на объект класса или на объект класса потомка. Переменная типа интерфейса может хранить ссылку на объект любого класса реализующего интерфейс. Перемененная типа массива может хранить ссылку на объект массива. Значением переменной ссылочного типа может быть специальное значение null которое не ссылается ни на какой объект. Значение null является значением по умолчанию для ссылочных типов. При объявлении значение ссылочной переменной может быть инициализировано.

Класс Object – ссылочный тип который является предком любого другого класса или массива. Переменная типа класса Object может содержать либо ссылку null, либо ссылку на любой объект.

Создание и использование объектов

Простой ссылочный тип

```
class Employee {

   Employee(String name, int salary) {
      this.name = name;
      this.salary = salary;
   }

   public void raiseSalary(int byPercent) {
      int raise = salary * byPercent / 100;
      salary += raise;
   }

   int salary;
   String name;
}
```

Создание объектов

```
public class NewDemo {

   public static void main(String[] args) {

      Employee bob;
      bob = new Employee("Robert", 20000);

      Employee alice = new Employee("Alice", 10000);

      System.out.println("Object employee: name = " + bob.name + ", salary = "
            + bob.salary);
      System.out.println("Object employee: name = " + alice.name + ", salary = "
            + alice.salary);
   }
}
```

```
Object employee: name = Robert, salary = 20000
Object employee: name = Alice, salary = 10000
```

Операторы

Операторы

Операторы == и != могут применяться к паре переменных ссылочных типов. Оператор == возвращает true если две переменные ссылаются на один объект и false если не ссылаются. Оператор != возвращает true если две ссылочные переменные не ссылаются на один объект и false если ссылаются.

Оператор = может применяться для того чтобы присвоить значение ссылки ссылочной переменной.

Сравнение ссылок

```
public class RefEqualityDemo {

   public static void main(String[] args) {

      Employee bob = new Employee("Robert", 20000);
      Employee robert = new Employee("Robert", 20000);

      System.out
            .println("Names and salaries are equal: "
                  + ((robert.name.equals(bob.name)) && (robert.salary == bob.salary)));

      System.out.println("References are equal: " + (robert == bob));
   }
}
```

```
Names and salaries are equal: true
References are equal: false
```

Присваивание и сравнение ссылок

```
public class AssignDemo {

   public static void main(String[] args) {

      Employee bob = new Employee("Robert", 20000);
      Employee robert = bob;

      System.out
            .println("Names and salaries are equal: "
                  + ((robert.name.equals(bob.name)) && (robert.salary == bob.salary)));

      System.out.println("References are equal: " + (robert == bob));
   }
}
```

```
Names and salaries are equal: true
References are equal: true
```

Изменение значения используя другую ссылку

Изменение значения используя другую ссылку

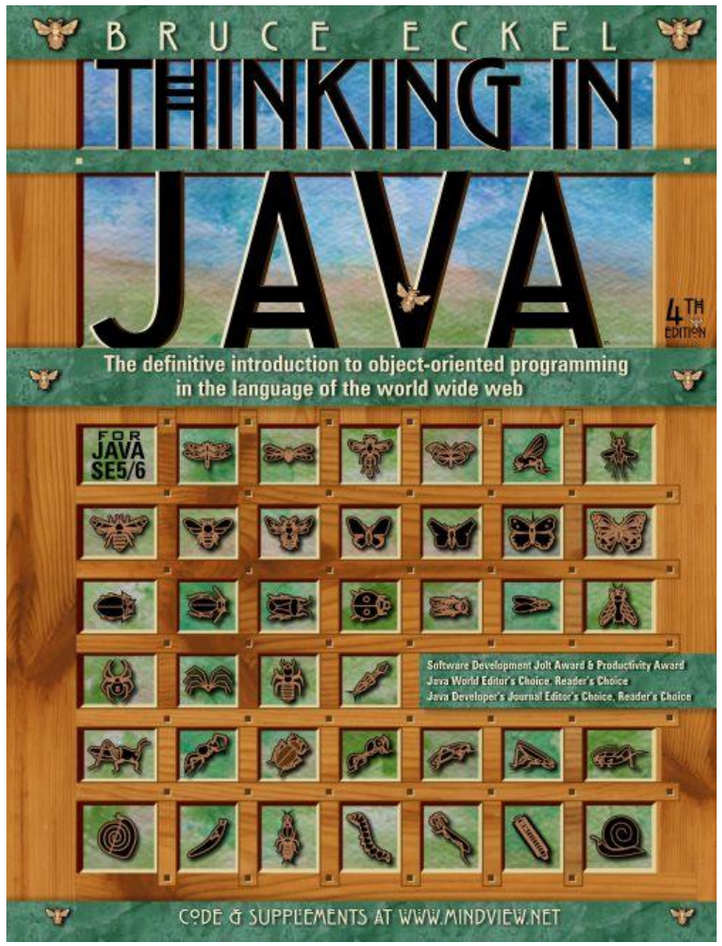
```
public class ChangeRefDemo {

   public static void main(String[] args) {

      Employee bob = new Employee("Robert", 20000);
      Employee robert = bob;

      System.out.println("Object employee: name = " + bob.name + ", salary = "
            + bob.salary);

      System.out.println("Doubling salary ...");

      robert.raiseSalary(100);

      System.out.println("Object employee: name = " + bob.name + ", salary = "
            + bob.salary);

   }
}
```

```
Object employee: name = Robert, salary = 20000
Doubling salary ...
Object employee: name = Robert, salary = 40000
```

Ссылочные типы как параметры

Передача по ссылке или по значению?

Авторы книг по Java продолжают спорить используется ли в Java передача параметров по ссылке или по значению. Можно говорить что используется передача объектов по ссылке или что используется передача ссылок по значению. Когда в качестве параметра в метод передаётся значение ссылочной переменной метод получает копию ссылки на объект и может изменить состояние объекта но метод не может изменить ссылку в исходной ссылочной переменной.



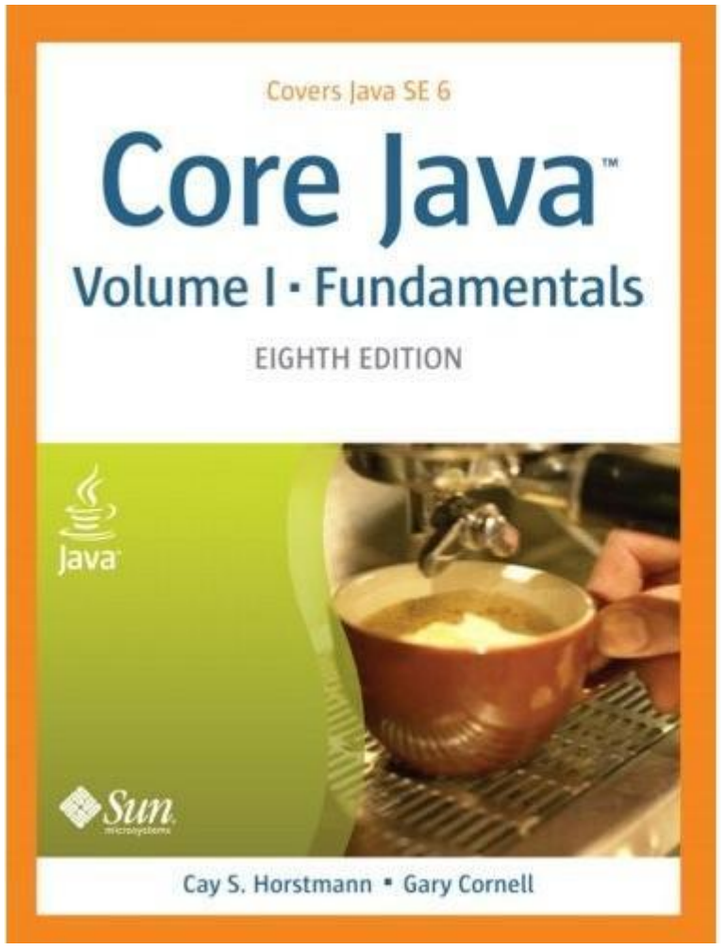
You manipulate objects with references

Each programming language has its own means of manipulating elements in memory. Sometimes the programmer must be constantly aware of what type of manipulation is going on. Are you manipulating the element directly, or are you dealing with some kind of indirect representation (a pointer in C or C++) that must be treated with a special syntax?

All this is simplified in Java. You treat everything as an object, using a single consistent syntax. Although you *treat* everything as an object, the identifier you manipulate is actually a "reference" to an object.[1] You might imagine a television (the object) and a remote control (the reference). As long as you're holding this reference, you have a connection to the television, but when someone says, "Change the channel" or "Lower the volume," what you're manipulating is the reference, which in turn modifies the object. If you want to move around

---

[1] This can be a flashpoint. There are those who say, "Clearly, it's a pointer," but this presumes an underlying implementation. Also, Java references are much more akin to C++ references than to pointers in their syntax. In the 1st edition of this book, I chose to invent a new term, "handle," because C++ references and Java references have some important differences. I was coming out of C++ and did not want to confuse the C++ programmers whom I assumed would be the largest audience for Java. In the 2nd edition, I decided that "reference" was the more commonly used term, and that anyone changing from C++ would have a lot more to cope with than the terminology of references, so they might as well jump in with both feet. However, there are people who disagree even with the term "reference." I read in one book where it was "completely wrong to say that Java supports pass by reference," because Java object identifiers (according to that author) are *actually* "object references." And (he goes on) everything is *actually* pass by value. So you're not passing by reference, you're "passing an object reference by value." One could argue for the precision of such convoluted explanations, but I think my approach simplifies the understanding of the concept without hurting anything (well, the language lawyers may claim that I'm lying to you, but I'll say that I'm providing an appropriate abstraction).



Many programming languages (in particular, C++ and Pascal) have two methods for parameter passing: call by value and call by reference. Some programmers (and unfortunately even some book authors) claim that the Java programming language uses call by reference for objects. However, that is false. Because this is such a common misunderstanding, it is worth examining a counterexample in detail.

Изменение ссылки

```
public class SwapDemo {

   public static void main(String[] args) {

      Employee alice = new Employee("Alice", 20000);
      Employee bob = new Employee("Robert", 30000);

      System.out.println("Employee Alice: name = " + alice.name + ", salary = " + alice.salary);
      System.out.println("Employee Bob: name = " + bob.name + ", salary = " + bob.salary);

      Employee temp = alice;
      alice = bob;
      bob = temp;

      System.out.println("Employee Alice: name = " + alice.name + ", salary = " + alice.salary);
      System.out.println("Employee Bob: name = " + bob.name + ", salary = " + bob.salary);
   }
}
```

```
Employee Alice: name = Alice, salary = 20000
Employee Bob: name = Robert, salary = 30000
Employee Alice: name = Robert, salary = 30000
Employee Bob: name = Alice, salary = 20000
```

Передача ссылки по значению

```
public class NoSwapDemo {

   public static void main(String[] args) {

      Employee alice = new Employee("Alice", 20000);
      Employee bob = new Employee("Robert", 30000);

      System.out.println("Employee Alice: name = " + alice.name + ", salary = " + alice.salary);
      System.out.println("Employee Bob: name = " + bob.name + ", salary = " + bob.salary);

      swap(alice, bob);

      System.out.println("Employee Alice: name = " + alice.name + ", salary = " + alice.salary);
      System.out.println("Employee Bob: name = " + bob.name + ", salary = " + bob.salary);
   }

   public static void swap(Employee a, Employee b) {

      Employee temp;

      temp = a;
      a = b;
      b = temp;
   }
}
```

```
Employee Alice: name = Alice, salary = 20000
Employee Bob: name = Robert, salary = 30000
Employee Alice: name = Alice, salary = 20000
Employee Bob: name = Robert, salary = 30000
```

Изменение с помощью ссылки

```
public class YesChangeDemo {

   public static void main(String[] args) {

      Employee alice = new Employee("Alice", 20000);

      System.out.println("Employee Alice: name = " + alice.name + ", salary = "
            + alice.salary);

      DoubleSalary(alice);

      System.out.println("Employee Alice: name = " + alice.name + ", salary = "
            + alice.salary);
   }

   public static void DoubleSalary(Employee a) {

      a.raiseSalary(100);
   }
}
```

```
Employee Alice: name = Alice, salary = 20000
Employee Alice: name = Alice, salary = 40000
```