Содержание

ГЛАВА 6

Операторы и блоки

Оператор — это отдельная команда, осуществляющая конкретное действие, когда она выполняется интерпретатором Java:

```
GigSim simulator = new GigSim("Let's play guitar!");
```

Операторы в Java могут включать в себя выражения, блоки кода, передачу управления, обработку исключений, переменные, метки, утверждения, а также пустые, условные, итеративные и синхронизированные операторы. В операторах Java употребляются следующие зарезервированные слова: `if`, `else`, `switch`, `case`, `while`, `do`, `for`, `break`, `continue`, `return`, `synchronized`, `throw`, `try`, `catch`, `finally` и `assert`.

Операторы выражений

Оператор выражения — это такой оператор, который изменяет состояние программы. Любое выражение в Java оканчивается точкой с запятой. К числу операторов выражений относятся: присваивания, префиксные и постфиксные приращения (инкременты), префиксные и постфиксные вычитания (декременты), создание объектов и вызовы методов. Ниже приведены некоторые примеры операторов выражений.

```
isWithinOperatingHours = true;
++fret; patron++; --glassOfWater; pick--;
```

```
Guitarist guitarist = new Guitarist();
guitarist.placeCapo(guitar, capo, fret);
```

Пустой оператор

Пустой оператор не выполняет никакого действия и обозначается точкой с запятой (`;`) или фигурными скобками `{}` как пустой блок.

Блоки

Группа операторов называется просто *блоком* или же *блоком операторов*. Блок операторов заключается в фигурные скобки. Переменные и классы, объявленные в блоке, называются локальными переменными и локальными классами соответственно. Область видимости локальных переменных и локальных классов определяется тем блоком, в котором они объявлены.

Операторы в блоках интерпретируются в той последовательности, в которой они написаны, или в порядке управления выполнением программы. Ниже приведен пример блока операторов.

```
static {
  GigSimProperties.setFirstFestivalActive(true);
  System.out.println("First festival has begun");
  gigsimLogger.info("Simulator started 1st festival");
}
```

Условные операторы

Условные операторы `if`, `if else` и `if else if` относятся к категории операторов выбора дальнейшего выполнения программы. Они служат для выполнения других операторов по заданным условиям. Выражение для любого из этих операторов

должно иметь тип Boolean или boolean. Если это тип Boolean, то выполняется распаковка, т.е. автоматическое преобразование типа Boolean в тип boolean.

Условный оператор if

Условный оператор if состоит из выражения и единственного оператора или целого блока операторов, которые выполняются, если в результате вычисления условного выражения получается логическое значение true.

```
Guitar guitar = new Guitar();
guitar.addProblemItem("Треснувший гриф");
if (guitar.isBroken()) {
  Luthier luthier = new Luthier();
  luthier.repairGuitar(guitar);
}
```

Условный оператор if else

Если условный оператор if дополняется оператором else, то первый блок операторов в конструкции if else выполняется в том случае, если в результате вычисления условного выражения получается логическое значение true. В противном случае выполняется второй блок операторов в конструкции if else.

```
CoffeeShop coffeeshop = new CoffeeShop();
if (coffeeshop.getPatronCount() > 5) {
  System.out.println("Play the event.");
} else {
  System.out.println("Go home without pay.");
}
```

Условный оператор if else if

Условный оператор if else if обычно применяется в том случае, когда требуется сделать выбор среди нескольких блоков кода. Если же заданные критерии выбора не позволяют

выполнить ни один из этих блоков, то выполняется код в последнем блоке оператора `else`.

```
ArrayList<Song> playList = new ArrayList<>();
Song song1 = new Song("Mister Sandman");
Song song2 = new Song("Amazing Grace");
playList.add(song1);
playList.add(song2);
...
int numOfSongs = playList.size();
if (numOfSongs <= 24) {
  System.out.println("Do not book");
} else if ((numOfSongs > 24) & (numOfSongs < 50)){
  System.out.println("Book for one night");
} else if ((numOfSongs >= 50)) {
  System.out.println("Book for two nights");
} else {
  System.out.println("Book for the week");
}
```

Оператор `switch`

Оператор `switch` является управляющим, передавая управление одному из следующих за ним операторов `case` в зависимости от значения заданного в нем выражения. Оператор `switch` способен оперировать данными типа `char`, `byte`, `short`, `int`, оболочками типа `Character`, `Byte`, `Short` и `Integer`, перечислениями и символьными строками (в виде объектов типа `String`). Поддержка объектов `String` была внедрена только в версии Java SE 7. Оператор `break` служит для завершения работы оператора `switch`. Если в операторе `case` отсутствует оператор `break`, то выполняется следующая за ним строка кода (как правило, следующий оператор `case`).

И это продолжается до тех пор, пока не будет достигнут оператор `break` или конец оператора `switch`. Допускается единственная метка `default`, которая обычно указывается в конце оператора `switch` ради повышения удобочитаемости кода, как показано ниже.

```
String style;
String guitarist = "Eric Clapton";
...
switch (guitarist) {
  case "Chet Atkins":
    style = "Nashville sound";
    break;
  case "Thomas Emmanuel":
    style = "Complex fingerstyle";
    break;
  default:
    style = "Unknown";
    break;
}
```

Итеративные операторы

Операторы простого и расширенного цикла `for`, а также операторы `while` и `do-while` являются итерационными. Они служат для циклического выполнения определенных фрагментов кода.

Оператор цикла `for`

Оператор цикла `for` состоит из трех частей: инициализации, условного выражения и обновления. Как показано ниже, перед использованием этого оператора должна быть инициализирована переменная цикла (`i`). Условное выражение (`i < bArray.length()`) всегда вычисляется до начала выполнения цикла. Итерация (т.е. шаг цикла) начинается лишь в том случае, если результатом вычисления условного выражения оказывается истинное значение, а переменная цикла обновляется (`i++`) после каждой итерации.

```
Banjo [] bArray = new Banjo[2];
bArray[0] = new Banjo();
bArray[0].setManufacturer("Windsor");
bArray[1] = new Banjo();
bArray[1].setManufacturer("Gibson");
```

```
for (int i=0; i<bArray.length; i++){
  System.out.println(bArray[i].getManufacturer());
}
```

Оператор расширенного цикла `for`

Оператор расширенного цикла `for`, т.е. цикла типа `for in` или `for each`, служит для циклической обработки объекта или массива, допускающего перебор его элементов. Цикл выполняется однократно для каждого элемента массива или коллекции без применения счетчика, поскольку количество итераций известно заранее.

```
ElectricGuitar eGuitar1 = new ElectricGuitar();
eGuitar1.setName("Blackie");
ElectricGuitar eGuitar2 = new ElectricGuitar();
eGuitar2.setName("Lucille");
ArrayList <ElectricGuitar> eList = new ArrayList<>();
eList.add(eGuitar1); eList.add(eGuitar2);
for (ElectricGuitar e : eList) {
  System.out.println("Name:" + e.getName());
}
```

Оператор цикла `while`

В операторе цикла `while` сначала вычисляется условное выражение, и цикл выполняется лишь в том случае, если результатом вычисления условного выражения оказывается истинное значение. Выражение может быть типа `boolean` или `Boolean`.

```
int bandMembers = 5;
while (bandMembers > 3) {
  CoffeeShop c = new CoffeeShop();
  c.performGig(bandMembers);
  // задать произвольно от нуля до семи
  // членов музыкального коллектива
  bandMembers = new Random().nextInt(8);
}
```

Оператор цикла do-while

Цикл в операторе do-while всегда выполняется хотя бы один раз. И его выполнение будет продолжаться до тех пор, пока результатом вычисления условного выражения не окажется истинное значение. Условное выражение может быть типа boolean или Boolean.

```
int bandMembers = 1;
do {
  CoffeeShop c = new CoffeeShop();
  c.performGig(bandMembers);
  Random generator = new Random();
  bandMembers = generator.nextInt(7) + 1; // 1-7
} while
```

Передача управления

Операторы передачи управления служат для изменения хода выполнения программы. К их числу относятся операторы break, continue и return.

Оператор break

Оператор break без метки служит для выхода из текущей ветви оператора switch или немедленного выхода из цикла. Этот оператор способен прервать работу простого и расширенного цикла for, а также циклов типа while и do-while.

```
Song song = new Song("Pink Panther");
Guitar guitar = new Guitar();
int measure = 1; int lastMeasure = 10;
while (measure <= lastMeasure) {
  if (guitar.checkForBrokenStrings()) {
    break;
  }
  song.playMeasure(measure);
  measure++;
}
```

А оператор `break` с меткой прерывает выполнение оператора цикла, следующего сразу за этой меткой. Метки обычно применяются во вложенных циклах `for` и `while`, когда требуется точно определить, из какого именно цикла следует выйти. Чтобы пометить какой-нибудь цикл или оператор, достаточно ввести оператор с меткой непосредственно перед помечаемым циклом или оператором, как показано в приведенном ниже примере кода.

```
...
playMeasures:
while (isWithinOperatingHours()) {
  while (measure <= lastMeasure) {
    if (guitar.checkForBrokenStrings()) {
      break playMeasures;
    }
    song.playMeasure(measure);
    measure++;
  }
} // выход из цикла здесь
```

Оператор `continue`

Выполнение оператора `continue` без метки приводит к прекращению текущего шага простого или расширенного цикла `for`, а также цикла `while` или `do-while` и началу следующего шага соответствующего цикла. При этом будет выполнена проверка условий выполнения цикла. А оператор `continue` с меткой приводит к выполнению шага цикла, следующего сразу же после метки.

```
for (int i=0; i<25; i++) {
  if (playList.get(i).isPlayed()) {
    continue;
  } else {
    song.playAllMeasures();
  }
}
```

Оператор `return`

Оператор `return` служит для выхода из метода и возврата значения, если в объявлении метода указано, что данный метод должен возвратить значение определенного типа.

```
private int numberOfFrets = 18; // по умолчанию
...
public int getNumberOfFrets() {
  return numberOfFrets;
}
```

Оператор `return` указывать необязательно, если он является последним в теле метода, который ничего не возвращает.

Оператор `synchronized`

Ключевое слово `synchronized` может использоваться в Java для предоставления доступа к определенным участкам кода (например, к отдельным методам) только в одном потоке исполнения. Оператор `synchronized` позволяет управлять доступом к ресурсам, которые совместно используются в нескольких потоках исполнения. Подробнее об этом — в главе 14.

Оператор `assert`

Утверждения являются логическими выражениями, которые используются в режиме отладки кода с целью проверить, ведет ли себя код именно так, как и ожидалось. Они активизируются интерпретатором Java при запуске приложения и командной строки с параметром **`-enableassertions`** или **`-ea`**. Утверждения обозначаются следующим образом:

```
assert логическое_выражение;
```

Утверждения облегчают выявление ошибок, в том числе обнаружение неожиданных значений. Они предназначены для подтверждения предположений, которые всегда должны быть истинны (`true`). Если при выполнении прикладного кода

в режиме отладки утверждение окажется ложным (false), то генерируется исключение типа java.lang.AssertionError и происходит выход из программы. В противном случае ничего не происходит.

```
// значение переменной strings должно быть равно
// 4, 5, 6, 7, 8 или 12
assert (strings == 12 ||
       (strings >= 4 && strings <= 8));
```

Утверждения следует активизировать явным образом. Аргументы командной строки, предназначенные для активизации утверждений, описаны в главе 10.

В утверждении можно также указать необязательный код ошибки, как показано ниже. Несмотря на такое название, код ошибки на самом деле является обычным текстом или значением, используемым исключительно в информативных целях.

```
assert boolean_expression : errorcode;
```

Если утверждение, в котором содержится код ошибки, оказывается ложным, значение кода ошибки преобразуется в символьную строку и отображается для пользователя непосредственно перед выходом из программы. Ниже приведен пример утверждения, в котором используется код ошибки.

```
// показать неверное количество струн музыкального
// инструмента в переменной strings
assert (strings == 12 ||
       (strings >= 4 && strings <= 8)) :
       "Invalid string count: " + strings;
```

Операторы обработки исключений

Операторы обработки исключений служат для указания кода, который должен быть выполнен в исключительных ситуациях. Для обработки исключений предназначены ключевые слова throw и try/catch/finally, обозначающие соответствующие операторы. Подробнее об обработке исключений речь пойдет в главе 7.