ГЛАВА 6

Операторы и блоки

Оператор представляет собой отдельно взятую команду, которая осуществляет то или иное действие интерпретатора Java при выполнении программы.

```
GigSim simulator = new GigSim(
                                "Давай, сыграем на гитаре!");
```

К числу операторов Java относятся: выражение, пустой оператор, блок, условный оператор, итерация, передача управления, обработка исключения, переменная, метка, утверждение и синхронизированные операторы.

Зарезервированными словами, которые используются в операторах Java, являются: `if`, `else`, `switch`, `case`, `while`, `do`, `for`, `break`, `continue`, `return`, `synchronized`, `throw`, `try`, `catch`, `finally` и `assert`.

Операторы выражений

Оператор выражения — это оператор, который изменяет состояние программы; любое выражение в Java заканчивается точкой с запятой. К числу операторов выражения относятся: присвоения, префиксные и постфиксные приращения (инкременты), префиксные и постфиксные вычитания (декременты), создание объекта и вызовы методов. Ниже приведены примеры операторов выражений.

```
isWithinOperatingHours = true;
++fret; patron++; --glassOfWater; pick--;
Guitarist guitarist = new Guitarist();
guitarist.placeCapo(guitar, capo, fret);
```

Пустой оператор

Пустой оператор не выполняет какого-либо действия и записывается в виде символа точки с запятой (`;`) или как пустой блок `{}`.

Блоки

Группа операторов называется блоком или блоком операторов. Блок операторов заключается в фигурные скобки. Переменные и классы, объявленные в блоке, называются локальными переменными и локальными классами соответственно. Областью действия локальных переменных и локальных классов является блок, в котором они объявлены.

В блоках операторы интерпретируются друг за другом, в той последовательности, в которой они записаны, или в порядке управления выполнением программы. Ниже приведен пример блока.

```
static {
   GigSimProperties.setFirstFestivalActive(true);
   System.out.println("Первый фестиваль открыт!");
   gigsimLogger.info("Начался первый фестиваль.");
}
```

Условные операторы

Операторы `if`, `if else` и `if else if` являются операторами управления выполнением программы (операторами принятия решений). Они используются для выполнения операторов в зависимости от наступления тех или иных условий. Выражение для любого из этих операторов должно иметь тип `Boolean` или `boolean`. При использовании типа `Boolean` будет выполнена распаковка, т.е. автоматическое преобразование `Boolean` в `boolean`.

Оператор `if`

Оператор `if` состоит из выражения и оператора или блока операторов, которые выполняются, если значение соответствующего выражения равно `true`.

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

Оператор `if else`

Если оператор `if` используется в сочетании с `else`, то первый блок операторов выполняется, если значение соответствующего выражения равно `true`; в противном случае выполняется блок кода в `else`.

```
CoffeeShop coffeeshop = new CoffeeShop();
if (coffeeshop.getPatronCount() > 5) {
   System.out.println("Отобразить приветствие.");
} else {
   System.out.println("Автомат неисправен!");
}
```

Оператор `if else if`

Оператор `if else if` обычно используется, когда нужно сделать выбор между несколькими блоками кода. Когда используемый критерий выбора не позволяет выполнить ни один из этих блоков, выполняется код в последнем блоке `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("Мало песен.");
} else if ((numOfSongs >  24) & (numOfSongs < 50)){
   System.out.println("Хватит на один вечер");
} else if ((numOfSongs >= 50) & (numOfSongs < 100)) {
   System.out.println("Хватит на два вечера.");
} else {
   System.out.println("Хватит на неделю.");
}
```

Оператор `switch`

Оператор `switch` является оператором управления выполнением программы. В зависимости от значения его выражения управление передается одному из следующих за ним операторов `case`. Оператор `switch` работает с типами `char`, `byte`, `short`, `int`, а также с интерфейсными типами `Character`, `Byte`, `Short` и `Integer`; с типами перечислений и типом `String`. Поддержка объектов `String` была обеспечена только в Java SE 7. Оператор `break` используется для завершения работы оператора `switch`. Если оператор `case` не содержит оператора `break`, то будет выполняться следующая за ним строка кода (обычно следующий оператор `case`).

Это продолжается до тех пор, пока либо не будет достигнут оператор `break`, либо конец оператора `switch`. Допускается использование одной метки `default`; как правило, она используется в конце оператора `switch` с целью улучшения читабельности кода.

```
String style;
String guitarist = "Эрик Клэптон";
...
switch (guitarist) {
   case "Чет Аткинс":
      style = "Фингерстайл";
      break;

   case "Томми Эммануэль":
      style = "Сложная импровизация";
      break;
```

```
   default:
      style = "Неизвестен";
      break;
}
```

Итерационные операторы

Операторы простого и расширенного цикла `for`, а также операторы `while` и `do-while` являются итерационными. Они используются 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 (т.е. `i`). Условное выражение (т.е. `i < bArray.length()`) всегда вычисляется до начала выполнения цикла. Итерация (т.е. выполнение цикла) начинается лишь в случае, если значение условного выражения истинно, а переменная цикла обновляется (т.е. `i++`) после каждой очередной 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());
}
```

Операторы расширенного цикла `for`

Операторы расширенного цикла `for`, т.е. циклы типа “for in” или “for each”, используются для итерационной обработки того

или иного объекта или массива, допускающего перебор своих элементов. Цикл выполняется однократно для каждого элемента массива или коллекции и не использует счетчик, 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 выражение, и цикл выполняется лишь в том случае, если значение выражения истинно. Выражение может быть типа `boolean` или `Boolean`.

```
int bandMembers = 5;
while (bandMembers > 3) {
   CoffeeShop c = new CoffeeShop();
   c.performGig(bandMembers);
   Random generator = new Random();
   bandMembers = generator.nextInt(7) + 1; // 1-7
}
```

Оператор цикла `do-while`

В операторе `do-while` цикл всегда выполняется по меньшей мере один раз и будет выполняться до тех пор, пока значение выражения истинно. Само 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 (bandMembers > 3);
```

Передача управления

Операторы передачи управления используются для изменения потока команд в программе. К числу таких операторов относятся: `break`, `continue` и `return`.

Оператор `break`

Оператор `break` без метки используется для выхода из текущей ветки оператора `switch` или немедленного выхода из цикла. Этот оператор может прекращать 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++;
}
```

Оператор `break` с меткой приводит к завершению цикла с указанной меткой и передаче управления следующему за этим циклом оператору. Метки обычно используются с циклами `for` и `while`, когда есть вложенные циклы и требуется точно определить, из какого именно цикла нужно выйти. Чтобы пометить какой-либо цикл или оператор, поместите оператор метки непосредственно перед помечаемым циклом или оператором, как показано в приведенном ниже примере.

```
...
playMeasures:
while (isWithinOperatingHours()) {
   while (measure <= lastMeasure) {
      if (guitar.checkForBrokenStrings()) {
         break playMeasures;
      }
      song.playMeasure(measure);
      measure++;
   }
}
// Выход осуществляется в эту точку
```

Оператор `continue`

Выполнение оператора `continue` без метки приводит к прекращению выполнения текущей итерации простого или расширенного цикла `for`, а также цикла `while` или `do-while` и началу следующей итерации соответствующего цикла. При этом будет выполнена проверка условий цикла. Оператор `continue` с меткой вызывает следующую итерацию указанного цикла:

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

Оператор `return`

Оператор `return` используется для выхода из метода и возвращения того или иного значения, если в описании метода указано, что данный метод должен вернуть значение определенного типа.

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

Оператор `return` использовать необязательно, если он является последним оператором в методе и этот метод ничего не возвращает.

Оператор `synchronized`

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

Оператор `assert`

Утверждения (assertions) представляют собой булевы выражения, используемые в режиме отладки для проверки, ведет ли себя код именно так, как ожидалось (т.е. при запуске приложения с использованием параметра командной строки `-enableassertions` или `-ea` интерпретатора Java). Утверждения записываются следующим образом:

```
assert булево_выражение;
```

Утверждения облегчают выявление ошибок, в том числе обнаружение неожиданных значений. Они предназначены для подтверждения предположений, которые всегда должны быть истинны. При выполнении в режиме отладки, если значение утверждения ложно, генерируется исключение `java.lang.AssertionError` и осуществляется выход из программы; в противном случае ничего не происходит. Утверждения нужно активизировать в явном виде. Аргументы командной строки, используемые для активизации утверждений, описаны в главе 10.

```
// Значение переменной 'strings' должно
// равняться 4, 5, 6, 7, 8 или 12
```

```
assert (strings == 12 ||
       (strings >= 4 & strings <= 8));
```

В утверждении можно также указать необязательный код ошибки. Несмотря на такое название ("код ошибки"), в действительности он представляет собой обычный текст или значение, используемые исключительно для информационных целей.

```
assert булево_выражение : код_ошибки;
```

Когда утверждение, в котором содержится код ошибки, истинно, значение кода ошибки преобразуется в строку и отображается для пользователя непосредственно перед завершением работы. Ниже приведен пример утверждения, в котором используется код ошибки.

```
// Показать недопустимое количество струн
// для музыкального инструмента'
assert (strings == 12 ||
       (strings >= 4 && strings <= 8))
       : "Недопустимое число струн: " + strings;
```

Операторы обработки исключений

Операторы обработки исключений используются для указания кода, который должен быть выполнен при возникновении необычных обстоятельств. Для обработки исключений используются ключевые слова `throw` и `try/catch/finally`. Подробнее об обработке исключений можно прочитать в главе 7.