Правописание инструкции **assert**:

```
assert (boolexp): expression;
assert (boolexp);
```

Выражение **boolexp** может принимать только значение типов **boolean** или **Boolean**, а **expression** – любое значение, которое может быть преобразовано к строке. Если логическое выражение получает значение **false**, то генерируется исключение **AssertionError**, и выполнение программы прекращается с выводом на консоль значения выражения **expression** (если оно задано).

Механизм assertion хорошо подходит для проверки инвариантов, например, перечислений:

```
enum Mono { WHITE, BLACK }
…
String str = "WHITE";//"GRAY"
Mono mono = Mono.valueOf(str);
// …
   switch (mono) {
         case WHITE : // …
               break;
         case BLACK : // …
               break;
         default :
               assert false : "Colored!";
   }
```

Создателями языка не рекомендуется использовать assertion при проверке параметров **public**-методов. В таких ситуациях лучше генерировать исключения одного из типов: **IllegalArgumentException**, **NullPointerException** или собственное исключение. Нет также особого смысла в механизме assertion при проверке пограничных значений переменных, поскольку исключительные ситуации генерируются в этом случае без посторонней помощи.

Assertion можно включать для отдельных классов и пакетов при запуске виртуальной машины в виде:

```
java –enableassertions MyClass
```

или

```
java –ea MyClass
```

Для выключения применяется **–da** или **-disableassertions**.

Задания к главе 8

Вариант A

Выполнить задания на основе варианта A главы 4, контролируя состояние потоков ввода/вывода. При возникновении ошибок, связанных с корректностью выполнения математических операций, генерировать и обрабатывать исключительные ситуации. Предусмотреть обработку исключений, возникающих при нехватке памяти, отсутствии требуемой записи (объекта) в файле, недопустимом значении поля и т.д.

Вариант B

Выполнить задания из варианта B главы 4, реализуя собственные обработчики исключений и исключения ввода/вывода.