таких как **floor()**, **ceil()**, **rint()**, **round()**, **max()**, **min()**, которые выполняют задачи по округлению, поиску экстремальных значений, нахождению ближайшего целого и т.д. Рассмотрим пример обработки значения случайного числа, полученного с помощью метода **random()** класса **Math**.

```
/* пример # 11 : использование методов класса Math: MathMethods.java */
package chapt02;

public class MathMethods {
     public static void main(String[] args) {
          final int MAX_VALUE = 10;
          double d;
          d = Math.random() * MAX_VALUE;
          System.out.println("d = " + d);
          System.out.println("Округленное до целого ="
                    + Math.round(d));
          System.out.println("Ближайшее целое, "
                    + " <= исходного числа ="
                    + Math.floor(d));
          System.out.println("Ближайшее целое, "
                    + " >= исходного числа ="
                    + Math.ceil(d));
          System.out.println("Ближайшее целое значение"
                    + "к числу =" + Math.rint(d));
     }
}
```

Один из вариантов выполнения кода представлен ниже:

```
d = 0.08439575016076173
Округленное до целого =0
Ближайшее целое, <= исходного числа =0.0
Ближайшее целое, >= исходного числа =1.0
Ближайшее целое значение к числу =0.0
```

Управление приложением

Все приложения автоматически импортируют пакет **java.lang**. Этот пакет содержит класс **java.lang.System**, предназначенный для выполнения ряда системных действий по обслуживанию работающего приложения. Объект этого класса создать нельзя.

Данный класс, кроме полей **System.in**, **System.out** и **System.err**, предназначенных для ввода/вывода на консоль, содержит целый ряд полезных методов:

void gc() – запуск механизма «сборки мусора»;

void exit(int status) – прекращение работы виртуальной java-машины (JVM);

void setIn(InputStream in), **void setOut(PrintStream out)**, **void setErr(PrintStream err)** – переназначение стандартных потоков ввода/вывода;

Properties getProperties() – получение всех свойств;

String getProperty(String key) – получение значения конкретного свойства;

void setSecurityManager(SecurityManager s), **SecurityManager getSecurityManager()** – получение и установка системы безопасности;

void load(String filename) – запуск программы из ОС;

void loadLibrary(String libname) – загрузка библиотеки;

void arrayCopy(параметры**)** – копирование части одного массива в другой.

Управлять потоком выполнения приложения можно с помощью класса **java.lang.Runtime**. Объект класса **Runtime** создается при помощи вызова статического метода **getRuntime()**, возвращающего объект **Runtime**, который ассоциирован с данным приложением. Остановить виртуальную машину можно с помощью методов **exit(int status)** и **halt(int status)**. Существует несколько возможностей по очистке памяти: **gc()**, **runFinalization()** и др. Определить общий объем памяти и объем свободной памяти можно с помощью методов **totalMemory()** и **freeMemory()**.

```
/*пример # 12 : информация о состоянии оперативной памяти:
RuntimeDemo.java*/
package chapt02;

public class RuntimeDemo {
     public static void main(String[] args) {
          Runtime rt = Runtime.getRuntime();
          System.out.println("Полный объем памяти: "
                    + rt.totalMemory());
          System.out.println("Свободная память: "
                    + rt.freeMemory());
          double d[] = new double[10000];
          System.out.println("Свободная память после" +
             " объявления массива: " + rt.freeMemory());
             //инициализация процесса
          ProcessBuilder pb =
          new ProcessBuilder("mspaint","c:\\temp\\cow.gif");

          try {
                    pb.start(); // запуск mspaint.exe
          } catch (java.io.IOException e) {
                    System.err.println(e.getMessage());
          }
          System.out.println("Свободная память после "
          + "запуска mspaint.exe: " + rt.freeMemory());
             System.out.println("Список команд: "
                                   + pb.command());
     }
}
```

В результате выполнения этой программы может быть выведена, например, следующая информация:

```
Полный объем памяти: 2031616
Свободная память: 1903632
Свободная память после объявления массива: 1823336
Свободная память после запуска mspaint.exe: 1819680
Список команд: [mspaint, c:\temp\cow.gif]
```

В примере использованы возможности класса **java.lang.ProcessBuilder**, обеспечивающего запуск внешних приложений с помощью метода **start()**, в качестве параметров которого применяются строки с именем запускаемого приложения и загружаемого в него файла. Внешнее приложение использует для своей загрузки и выполнения память операционной системы.

Метод **arraycopy()** класса **System**, позволяет копировать часть одного массива в другой, начиная с указанной позиции.

```
/* пример # 13 : копирование массива: ArrayCopyDemo.java */
package chapt02;

public class ArrayCopyDemo {
     public static void main(String[] args) {
          int mas1[] = { 1, 2, 3 },
          mas2[] = { 4, 5, 6, 7, 8, 9 };
          show("mas1[]: ", mas1);
          show("mas2[]: ", mas2);
          // копирование массива mas1[] в mas2[]
          System.arraycopy(mas1, 0, mas2, 2, 3);
          /*
          0 – mas1[] копируется начиная с первого элемента,
          2 – элемент, с которого начинается замена,
          3 – количество копируемых элементов
           */
          System.out.printf("%n после arraycopy(): ");
          show("mas1[]: ", mas1);
          show("mas2[]: ", mas2);
     }
     private static void show(String s, int[] mas) {
          System.out.printf("%n%s", s);
          for (int i : mas) System.out.printf("%d ", i);
     }
}
```

В результате будет выведено:

```
mas1[]:  1 2 3
mas2[]:  4 5 6 7 8 9
после arraycopy():
mas1[]:  1 2 3
mas2[]:  4 5 1 2 3 9
```