Простейшая программа Hello, world!

```
// файл Hello.java //
public class Hello {
     public static void main(String[] args) {
          System.out.println("Привет, мир!");
     }
}
```

Обязательные требования:

1) Имя файла, содержащего код программы, должно совпадать с именем класса
2) Файл должен иметь расширение `.java`
3) Заглавные и строчные буквы различаются

Компиляция производится с помощью компилятора `javac`. При этом создается файл `Hello.class`, содержащий байт-код.

```
> javac Hello.java
```

Запуск программы осуществляется с помощью интерпретатора `java`. При этом расширение не указывается. При запуске управление передается методу `main`.

```
> java Hello
Привет, мир!
```

Передача аргументов с помощью командной строки

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          System.out.println("Привет, " + args[0] + "!");
     }
}
```

Массив `args` типа `String` содержит список аргументов командной строки. Аргументы разделяются пробелами. Элемент массива `args[0]` содержит первый аргумент командной строки после имени программы.

```
> javac Hello.java
> java Hello Вася
Привет, Вася!
```

Однако, данная программа не защищена от ошибок. Если при запуске ни одного аргумента не указать, то появится следующее сообщение:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at Hello.main(Hello.java:4)
```

Возникло исключение при попытке обратиться к элементу массива с несуществующим индексом.

Обработка исключений

Решить проблему можно с помощью обработки исключения

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          try {
               System.out.println("Привет, " + args[0] + "!");
          } catch (ArrayIndexOutOfBoundsException e) {
               System.out.println("Привет, мир!");
          }
     }
}
```

```
> javac Hello.java
> java Hello Вася
Привет, Вася!
> java Hello
Привет, мир!
```

Часть кода, где возможно возникновение исключений, заключается в блок `try`, после которого располагаются блоки `catch`, соответствующие исключениям, в которых происходит их обработка.

Компилятор следит за необходимостью обработки некоторых исключений, выдавая предупреждающие сообщения при отсутствии блока `try`. В основном это такие исключения, которые нельзя предотвратить. Например: не найден класс, достигнут конец файла, неверная кодировка, прерывание потока. Выход индекса за пределы массива относится к неконтролируемым исключениям (`RuntimeException`). Таких ситуаций можно избежать путем предварительной проверки, поэтому компилятор не требует обязательной их обработки.

Проверка на наличие аргументов

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          if (args.length > 0) {
               System.out.println("Привет, " + args[0] + "!");
          } else {
               System.out.println("Привет, мир!");
          }
     }
}
```

```
> javac Hello.java
> java Hello Вася
Привет, Вася!
> java Hello
Привет, мир!
```

В данном варианте обработка исключения заменена предварительной проверкой количества элеменов в массиве.

Использование множественного ветвления

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          switch (args.length) {
          case 1:
               System.out.println("Привет, "+args[0]+"!");
               break;
          case 0:
               System.out.println("Привет, мир!");
               break;
          case 2:
               System.out.println("Привет, "+args[0]+" и "+args[1]+"!");
               break;
          default:
               System.out.println("Привет всем!");
          }
     }
}
```

```
> javac Hello.java
> java Hello Вася
Привет, Вася!
> java Hello
Привет, мир!
> java Hello Вася Петя
Привет, Вася и Петя!
> java Hello Вася Петя Михалыч
Привет всем!
```

В данном варианте выводятся разные сообщения при разном количестве аргументов входной строки. Оператор `break` используется для выхода из блока `switch`. А вариант `default` обрабатывается, если ни одно значение `case` не подошло.

Цикл с постусловием

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          int i = 0;
          switch (args.length) {
          case 0:
               System.out.println("Привет, мир!");
               break;
          default:
               do {
                    System.out.println("Привет, "+args[i]+"!");
                    i++;
               } while (i < args.length);
          }
     }
}
```

```
> javac Hello.java
> java Hello
Привет, мир!
> java Hello Вася Петя Михалыч
Привет, Вася!
Привет, Петя!
Привет, Михалыч!
```

Так как условие выхода из цикла проверяется после тела цикла, то цикл выполнится хотя бы один раз, поэтому проверку количества аргументов производить надо.

Цикл с предусловием

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          int i = 0;
          switch (args.length) {
          case 0:
               System.out.println("Привет, мир!");
               break;
          default:
               while (i < args.length) {
                    System.out.println("Привет, "+args[i]+"!");
                    i++;
               }
          }
     }
}
```

```
> javac Hello.java
> java Hello
Привет, мир!
> java Hello Вася Петя Михалыч
Привет, Вася!
Привет, Петя!
Привет, Михалыч!
```

Если условие перед началом цикла не выполняется, то входа в цикл не будет.

Цикл for

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          for (int i = 0; i < args.length; i++) {
               System.out.println("Привет, "+args[i]+"!");
          }
     }
}
```

```
> javac Hello.java
> java Hello
> java Hello Вася Петя Михалыч
Привет, Вася!
Привет, Петя!
Привет, Михалыч!
```

Так как при нулевом количестве входных аргументов цикл не выполнится ни разу, то предварительную проверку можно исключить. В этом случае при отсутствии аргументов программа ничего не выведет.

Цикл по элементам массива

```
// файл Hello.java
public class Hello {
     public static void main(String[] args) {
          for (String s : args) {
               System.out.println("Привет, " + s + "!");
          }
     }
}
```

```
> javac Hello.java
> java Hello Вася Петя Михалыч
Привет, Вася!
Привет, Петя!
Привет, Михалыч!
```

Данная форма цикла `for` работает следующим образом. Переменной `s` присваивается очередной элемент массива `args` и для этого значения выполняется тело цикла.

Переменные и методы

```
// файл Hello.java
public class Hello {
     private String name;
     public static void main(String[] args) {
          for (String s : args) {
               name = s;
               say();
          }
     }
     public void say() {
          System.out.println("Привет, " + name + "!");
     }
}
```

Метод `main` используется как точка входа в программу. В нем обычно реализуется инициализация объектов и вызов их методов. Основное поведение объектов должно определяться в других методах. Поэтому вводим метод `say`, который и реализует основное действие нашего класса — вывод приветствия. Дополнительно определим переменную `name` типа `String`, в которой будет сохраняться очередной аргумент. Переменная объявлена с модификатором private, который разрешает доступ к ней только из текущего класса. Такой принцип, когда доступ к переменным внутри класса ограничен, а их изменение производится только с помощью общедоступных методов, называется инкапсуляцией.

Однако, данная программа отказывается компилироваться со следующими сообщениями:

```
Hello.java:6: non-static variable name cannot be referenced from a static
context
                         name = s;
                         ^
Hello.java:7: non-static method say() cannot be referenced from a static
context                  say();
                         ^
2 errors
```

Метод `main` является статическим. Статические переменные и методы принадлежат классу. А переменные и методы без модификатора `static` — конкретному объекту, поэтому при отсутствии объектов они не определены. Так как статический метод может вызываться без создания объектов, из статического метода запрещено обращение к нестатическим переменным и методам.

Использование методов (продолжение)

```
// файл Hello.java
public class Hello {
     private static String name;
     public static void main(String[] args) {
          for (String s : args) {
               name = s;
               say();
          }
     }
     public void static say() {
          System.out.println("Привет, " + name + "!");
     }
}
```

```
> javac Hello.java
> java Hello Вася Петя
Привет, Вася!
Привет, Петя!
```

Один из способов решить эту проблему — объявить метод `say()` и переменную `name` как `static`.

Использование объектов

```
// файл Hello.java
public class Hello {
     private String name;

     public Hello() {
          name = "мир";
     }

     public void setName(String s) {
          name = s;
     }

     public static void main(String[] args) {
          Hello h = new Hello();
          for (String s : args) {
               h.setName(s);
               h.say();
          }
     }
     public void say() {
          System.out.println("Привет, " + name + "!");
     }
}
```

```
> javac Hello.java
> java Hello Вася Петя
Привет, Вася!
Привет, Петя!
```

Предыдущий пример не использовал принципы объектно-ориентированного программирования. Исправляем этот недостаток. Убираем модификатор `static` у переменной `name` и метода `say()`, добавляем конструктор, который присваивает переменной name начальное значение, а также метод `setName(String)`, предназначенный для установки имени. В методе `main` теперь создается объект с помощью оператора `new`, и вызываются его методы.

Реализация основного класса

```
// файл Lab1.java
public class Lab1 {
     public static void main(String[] args) {
          Hello h = new Hello();
          for (String s : args) {
               h.setName(s);
               h.say();
          }
     }
}
// файл Hello.java
public class Hello {
     private String name;

     public Hello() {
          name = "мир";
     }
     public void setName(String s) {
          name = s;
     }
     public void say() {
          System.out.println("Привет, " + name + "!");
     }
}
```

```
> javac Lab1.java Hello.java
> java Lab1 Вася Петя
Привет, Вася!
Привет, Петя!
```

Перенесем метод `main` из класса `Hello` в основной класс `Lab1`, который будет предназначен для запуска приложения.

Наследование

```
// файл Lab1.java
public class Lab1 {
     public static void main(String[] args) {
          Morning h = new Morning();
          for (String s : args) {
               h.setName(s);
               h.say();
          }
     }
}
// файл Hello.java
public class Hello {
     protected String name;

     public Hello() {
          name = "мир";
     }
     public void setName(String s) {
          name = s;
     }
     public void say() {
          System.out.println("Привет, " + name + "!");
     }
}
// файл Morning.java
public class Morning extends Hello {
     public void say() {
          System.out.println("Доброе утро, " + name + "!");
     }
}
```

```
> javac Lab1.java Hello.java
> java Lab1 Вася Петя
Доброе утро, Вася!
Доброе утро, Петя!
```

Реализуем класс `Morning`, который наследуется от класса `Hello`. Это делается с помощью ключевого слова `extends`. Переменную name в классе `Hello` определяем как `protected`. Это позволяет получить доступ к ней не только из класса `Hello`, но из его потомков. Метод `setName` изменять не надо, а метод say должен работать по другому, поэтому мы его переопределяем.
Конструктор не наследуется, но при отсутствии конструктора в классе-потомке автоматически добавляется конструктор по умолчанию, который вызывает конструктор класса-предка.

Документирующие комментарии

```
// файл Lab1.java
/** Основной класс программы */
public class Lab1 {
     public static void main(String[] args) {
          Morning h = new Morning();
          for (String s : args) {
               h.setName(s);
               h.say();
          }
     }
}
// файл Hello.java
/** Класс для приветствия типа «Привет» */
public class Hello {
     protected String name;
/** Если имя было не задано,
    то будет использоваться слово «мир». */
     public Hello() {
          name = "мир";
     }
     public void setName(String s) {
          name = s;
     }
/** вывод приветствия */
     public void say() {
          System.out.println("Привет, " + name + "!");
     }
}
// файл Morning.java
/** Класс для приветствия типа «Доброе утро!» */
public class Morning extends Hello {
     public void say() {
          System.out.println("Доброе утро, " + name + "!");
     }
}
```

Документирующие комментарии имеют вид /** Комментарий (много строк) */
Для создания документации используется утилита javadoc

```
> javadoc *.java
```

Она генерирует пакет документации в стандартизованном виде в формате HTML.
При этом документирующие комментарии вставляются в документацию.
Посмотреть получившуюся документацию можно с помощью веб-браузера.

Задание для самостоятельного выполнения

Придумать еще один вариант программы Hello world, в котором реализовать какую-то дополнительную функцию.