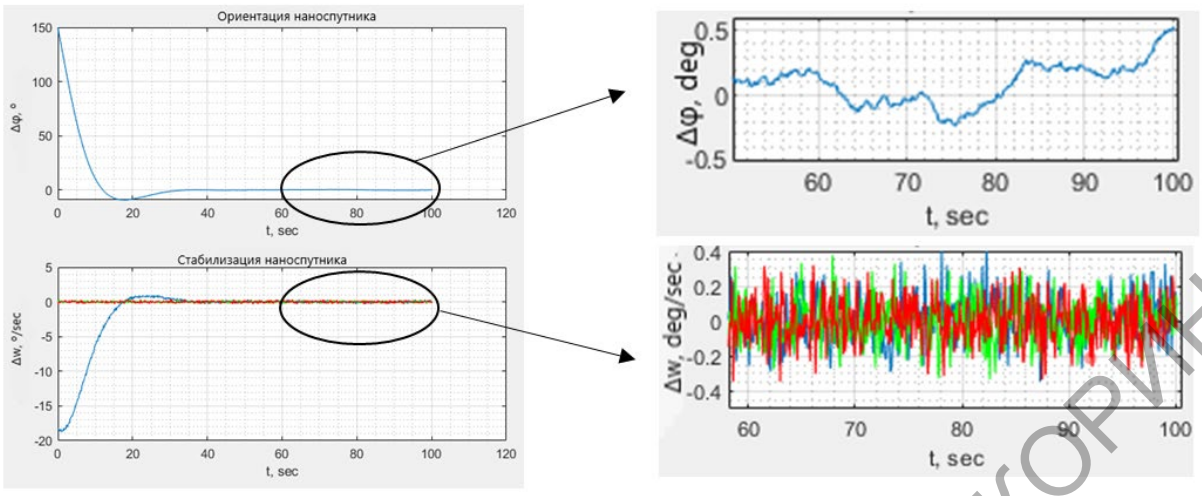

Рисунок 3 – Графики моделирования поворота и стабилизации макета

Литература

1. Васильев В. Н. Системы ориентации космических аппаратов / В. Н. Васильев. – М.: ФГУП «НПП ВНИИЭМ», 2009. – 310 с.

2. Spacecraft Dynamics and Control: The Embedded Model Control Approach/ Enrico Canuto, Carlo Novara, Donato Carlucci, Carlos Perez Montenegro, Luca Massotti. – 1st ed. – Butterworth-Heinemann, 2018. – 790 p.

А. Ю. Никонович
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **О. М. Дерюжкова**, канд. физ.-мат. наук, доцент

ЭКРАНИРОВАНИЕ СИМВОЛОВ В JAVA

Строка – это последовательность символов. Эти символы могут быть любыми буквами, цифрами, знаками препинания и так далее. Главное при создании строки – вся последовательность должна быть заключена в кавычки.

Но что делать, если нужно создать строку, которая сама должна содержать кавычки? Например:

```
public class main {
  public static void main(String[] args) {
   String myFavoriteBook = new String ("My favorite book is "Fire and blood " by George R.R. Martin");
```

```
  }
}
```

Кажется, что компилятор чем-то недоволен и выдаёт ошибку. На самом деле все очень просто. Компилятор интерпретирует кавычки очень специфическим образом, то есть ожидает, что в них будут заключены строки. И каждый раз, когда компилятор видит ", он ожидает, что за кавычками последует вторая кавычка, и что содержимое между ними будет текстом строки, которая будет создана компилятором. В данном случае кавычки вокруг фразы «Fire and Blood» заключены в другие кавычки. Когда компилятор достигает этого фрагмента текста, он просто не понимает, что должен делать. Кавычки предполагают, что строка должна быть создана. Но это то, что компилятор уже делает! И вот почему: попросту говоря, компилятор не понимает, что от него ожидается.

Для этого в Java используется экранирование символов. Это достигается с помощью специального символа: \. Данный символ обычно называют «обратной косой чертой». В Java, обратная косая черта в сочетании с символом, который нужно «экранировать», называется управляющей последовательностью. Например, \ **"** – это управляющая последовательность для отображения кавычек на экране. Обнаружив эту конструкцию в коде, компилятор поймет, что это всего лишь кавычка, которая должна отображаться на экране.

Попробуем изменить код:

```
public static void main(String[] args) {
    String myFavoriteBook = new String ("My favorite book is \"Fire and Blood\" by George R.R. Martin ");
    System.out.println(myFavoriteBook);
  }
}
```

Вывод в консоль:

My favorite book is “Fire and Blood” by George R.R. Martin

Кавычки ни в коем случае не единственные символы, которые нам нужно избегать:

```
public class main {
  public static void main(String[] args) {
    String workFiles= new String ("My work files are in D:\Work Projects\java");
    System.out.println(workFiles);
  }
}
```

Еще одна ошибка! И снова компилятор не понимает, что делать. Ведь компилятор не знает о символе \ ничего, кроме управляющей последовательности! Ожидается, что за обратной косой чертой будет следовать определенный символ, который он должен интерпретировать особым образом (например, кавычки). Но в этом случае за \ следует обычные буквы. Итак, компилятор снова запутался. Что необходимо делать? Точно так же, как и раньше: просто добавляем еще один символ \ к уже имеющемуся \.

```
public class Main {

  public static void main(String[] args) {

    String workFiles= new String ("My work files are in D:\\Work Projects\\java");
    System.out.println(workFiles);

  }
}
```

Посмотрим, что получится. Вывод в консоль:
My work file are in D: \ Work Projects \ java

Компилятор сразу определяет, что \ – обычные символы, которые должны отображаться вместе с остальными.

В Java довольно много управляющих последовательностей. Вот полный список:
\ t – таб;
\ b – backspace (шаг назад по тексту или удаление одного символа);
\ n – новая строка;
\ r – возврат каретки;
\ f – подача формы;
\ ' – одинарная кавычка;
\ " – двойная кавычка.

Таким образом, если компилятор встречает \ n в тексте, он понимает, что это не просто символ и буква для отображения на консоли, а, скорее, специальная команда «перейти на новую строку».

Литература

1. Bloch, J. Effective Java, 3rd edition / J. Bloch. – Addison-Wesley Professional, 2017. – 416 p.