pi-3. Javaプログラミングの基本：メソッドと抽象化によるプログラミング

（Java マスター講座：基礎から応用まで）

URL: https://www.kkaneko.jp/pro/pi/index.html

金子邦彦

CC BY NC SA

全体まとめ

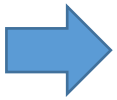
- **抽象化**は ：**変数**を使って，**複数の式**を**１つにまとめる**ことなど

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

- **メソッド呼び出し**：ジャンプ，変数の生成と消去が自動で行われる

アウトライン

番号	項目
	復習
3-1	式，変数
3-2	式の抽象化とメソッド
3-3	メソッド呼び出し

各自、資料を読み返したり、課題に取り組んだりも行う

この授業では、**Java** を用いて基礎を学び、マスターする

ソースコード (source code)

- **プログラム**を，何らかの**プログラミング言語**で書いたもの
- 「**ソフトウエアの設計図**」ということも.

人間も読み書き，編集できる

```
public class YourClassNameHere {
    public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println(x + y);
    }
}
```

100 + 200 を計算する Java 言語プログラム

オブジェクトとメソッド

hero.moveDown()

hero　　オブジェクト
moveDown()　メソッド
間を「.」で**区切って**いる

- **メソッド: オブジェクト**に属する操作や処理.
- **メソッド**呼び出しでは，**引数**を指定することがある．**引数**（ひきすう）は，**メソッド**に渡す値のこと

hero.attack("fence", 36, 26)

Java プログラムの書き方

プログラムの例

```
x = 100
a = x + 200
enermy1 = hero.findNearestEnemy()
hero.attack(enemy1)
```

- **代入**：**オブジェクト名** ＋ 「**=**」
  ＋ 式または値またはメソッド呼び出し
- **メソッドアクセス**：**オブジェクト名** ＋ 「**.**」
  ＋ **メソッド名** ＋「**()**」 （引数を付けることも）

その他，属性アクセス，関数呼び出し，制御，「*」，「+」などの演算子，コマンド，定義など

Java Tutor の起動

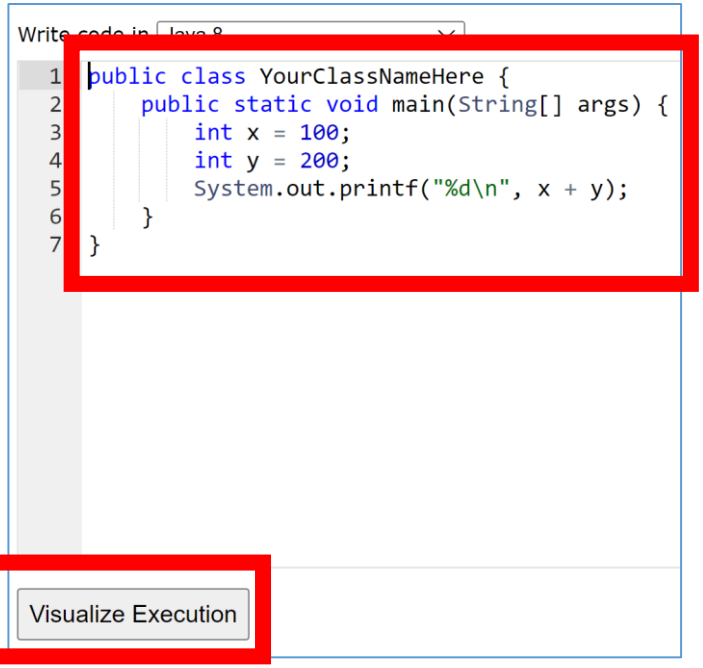
① **ウェブブラウザ**を起動する

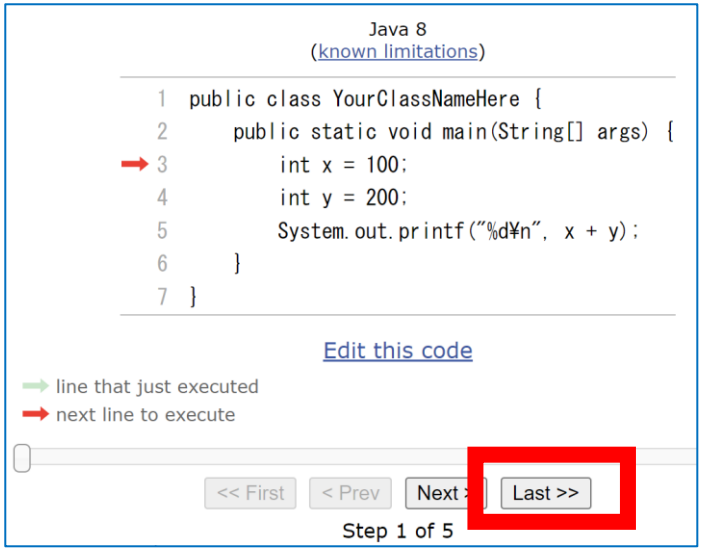
② **Java Tutor** を使いたいので，次の URL を開く

https://pythontutor.com/java.html

③ 「**Java**」をクリック　⇒ **編集画面**が開く

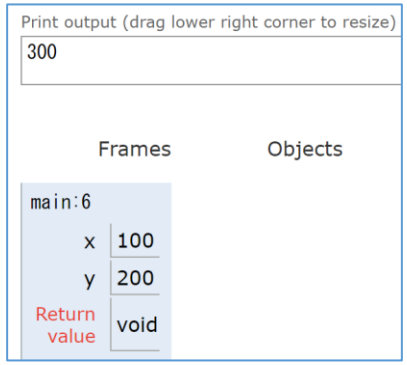
Java Tutor でのプログラム実行手順

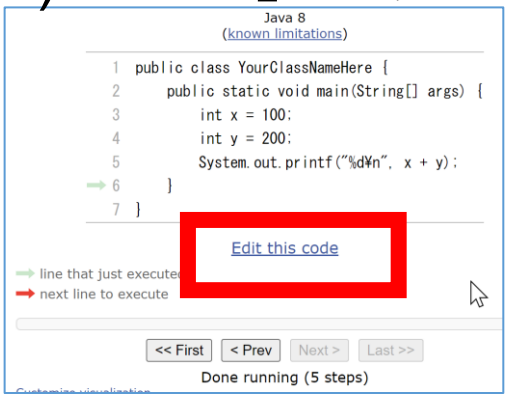




(1)「**Visualize Execution**」をクリックして**実行画面**に切り替える

(2)「**Last**」をクリック.





(3) **<u>実行結果を確認</u>**する.

(4)「**Edit this code**」をクリックして**編集画面**に戻る

Java Tutor 使用上の注意点①

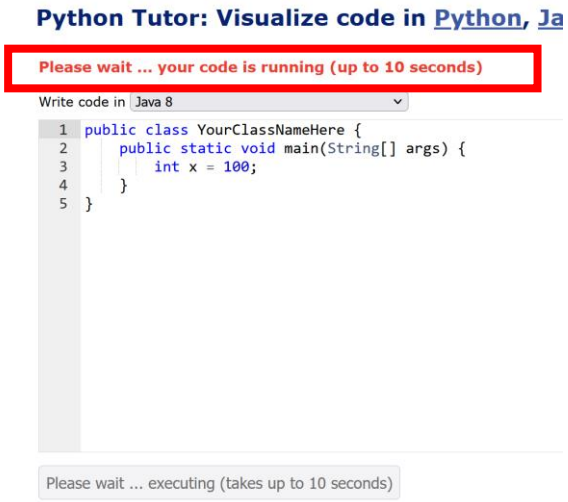
- **実行画面で，次のような赤の表示が出ることがある → 無視してよい**

  過去の文法ミスに関する確認表示

  邪魔なときは「**Close**」

Java Tutor 使用上の注意点②

「**please wait ... executing**」のとき，**１０秒ほど待つ**.



→ **混雑しているとき**は，**「Server Busy・・・」というメッセージが出る**ことがある.

混雑している．**少し（数秒から数十秒）待つ**と自動で表示が変わる（変わらない場合には，操作をもう一度行ってみる）

3-1. 式，变数

式の実行結果

式の実行結果として，**値が得られる**

```
1  public class YourClassNameHere {
2      public static void main(String[] args) {
3          System.out.println(100 * 200);
4      }
5  }
```

Print output (drag lowe

20000

プログラム

実行結果

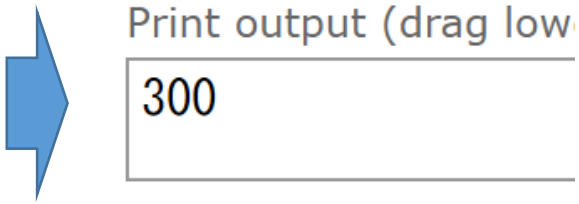
変数

- **名前の付いたオブジェクト**のことを**変数**という場合がある（数学の変数とは違う意味）

- **式**の中に**変数**を**含めること**ができる

```
1  public class YourClassNameHere {
2      public static void main(String[] args) {
3          int x = 100;
4          int y = 200;
5          System.out.println(x + y);
6      }
7  }
```

プログラム



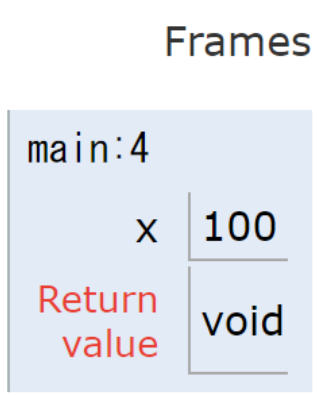
実行結果

代入

- **代入**：プログラムで，「**x = 100**」のように書くと，**x の値が 100 に変化**する

```
1  public class YourClassNameHere {
2      public static void main(String[] args) {
3          int x = 100;
4      }
5  }
```

プログラム



実行結果

演習

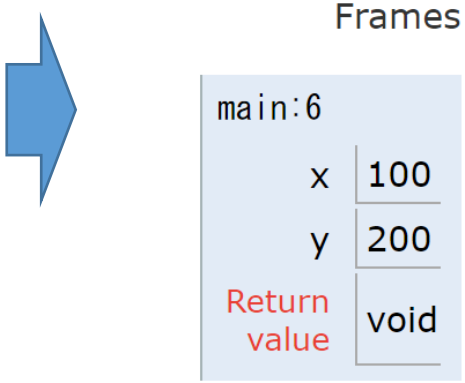
資料：**16 ～ 19**

【トピックス】

- **式と変数**

① Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する

```
public class YourClassNameHere {
    public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println(x + y);
    }
}
```

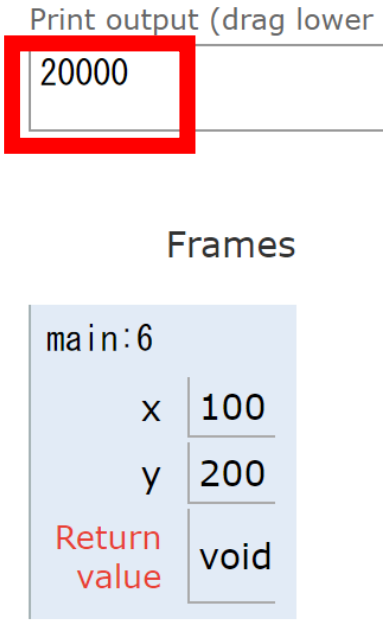


結果の「**300**」を確認

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

② Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する

```
public class YourClassNameHere {
    public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println(x * y);
    }
}
```

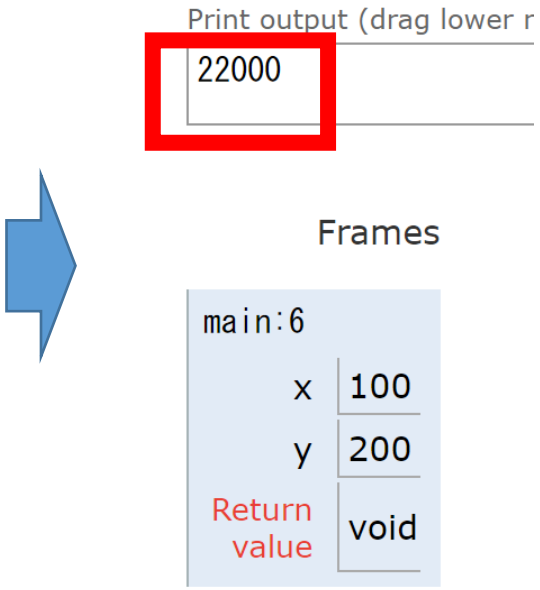


結果の
「**20000**」を確認

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

③ Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する

```
public class YourClassNameHere {
    public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println((x + 10) * y);
    }
}
```



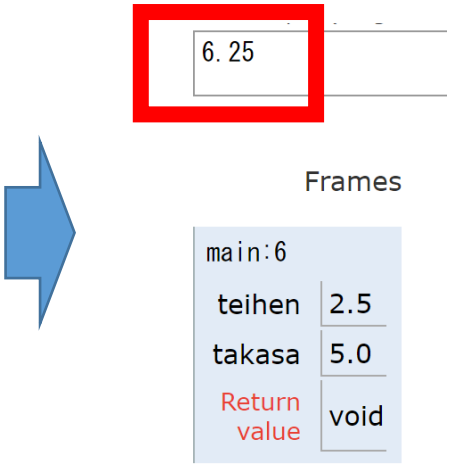
結果の「**22000**」を確認

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

④ Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する

底辺が**2．5**で，高さが**5**のとき，
三角形の面積は，面積：**6．25**

```
public class YourClassNameHere {
    public static void main(String[] args) {
        double teihen = 2.5;
        double takasa = 5;
        System.out.println(teihen * takasa / 2);
    }
}
```



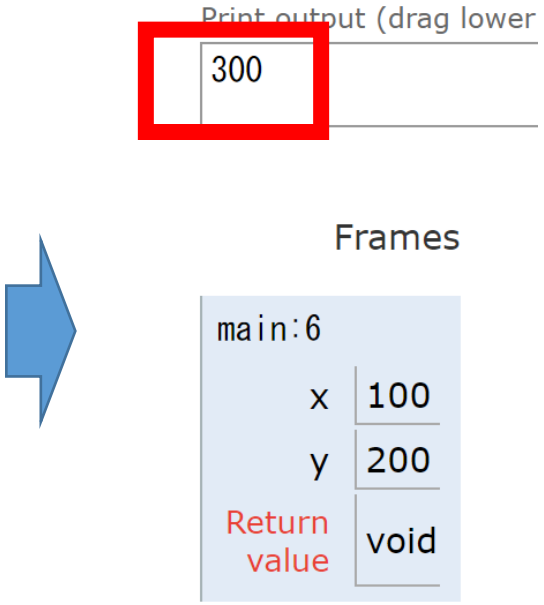
結果の
「**6.25**」を確認

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

まとめ

- **代入**：プログラムで，「**x = 100**」のように書くと，**x の値が 100 に変化**する
- **式**の実行結果として，**値が得られる**
- **式**の中に，**変数名**を書くことができる

```
public class YourClassNameHere {
    public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println(x + y);
    }
}
```



3-2. 式の抽象化とメソッド

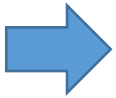
式の抽象化

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

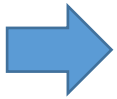
式の抽象化とメソッド

100 * 1.1

150 * 1.1

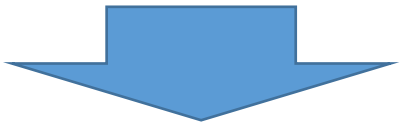
400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）



```
1  public class YourClassNameHere {
2      public static double foo(double a) {
3          return a * 1.1;
4      }
5      public static void main(String[] args) {
6          System.out.println(foo(100));
7          System.out.println(foo(150));
8          System.out.println(foo(400));
9      }
10 }
```

メソッド foo を使用

式「**a * 1.1**」を含む**メソッド foo** を**定義**

Print output (drag lower rig

```
110.00000000000001
165.0
440.00000000000006
```

実行結果

メソッド

```
public static double foo(double a) {
  return a * 1.1;
}
```

- このメソッドの本体は
  「**return a * 1.1;**」

- このメソッドは，式「**a * 1.1**」に，**名前 foo** を付けたものと考えることもできる

式の抽象化とメソッド

抽象化前

```
1  public class YourClassNameHere {
2      public static void main(String[] args) {
3          System.out.println(100 * 1.1);
4          System.out.println(150 * 1.1);
5          System.out.println(400 * 1.1);
6      }
7  }
```

類似した複数の**式**

抽象化後

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.println(foo(100));
        System.out.println(foo(150));
        System.out.println(foo(400));
    }
}
```

メソッドの定義と使用

Print output (drag lower rig

```
110.00000000000001
165.0
440.00000000000006
```

実行結果

Print output (drag lower rig

```
110.00000000000001
165.0
440.00000000000006
```

同じ実行結果になる

演習

資料：**27**

【トピックス】

- **式の抽象化とメソッド**

Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する（あとで使うので消さないこと）

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.println(foo(100));
        System.out.println(foo(150));
        System.out.println(foo(400));
    }
}
```

Print output (drag lower rig

```
110.00000000000001
165.0
440.00000000000006
```

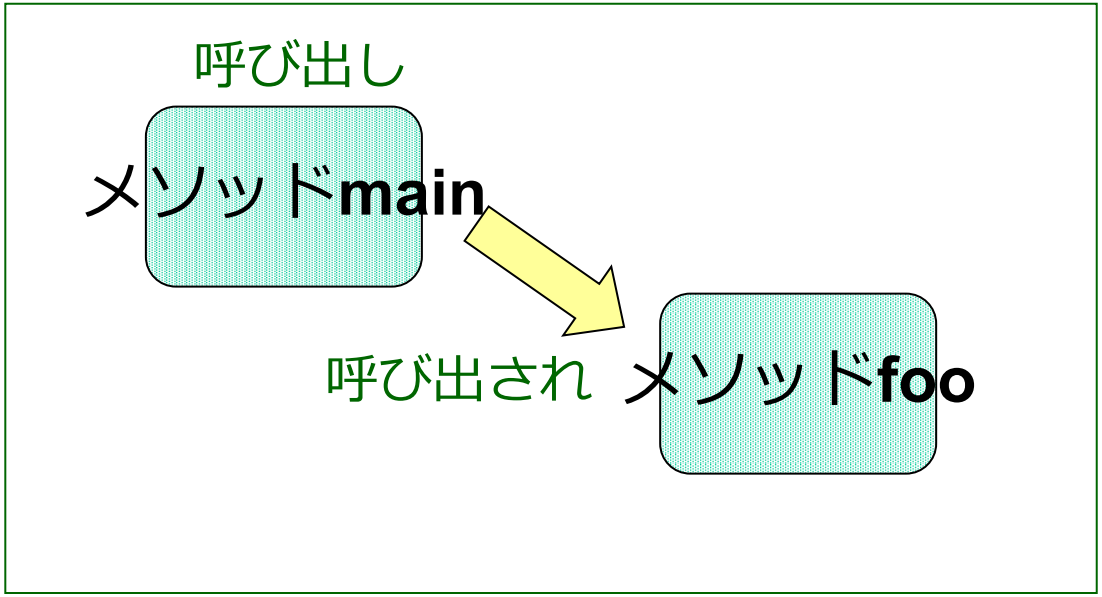
実行結果を確認

Visualize Execution をクリック，Last をクリック

※ 編集画面に戻るには Edit this code をクリック

3-3. メソッド呼び出し

メソッド呼び出し



ステップ実行

ステップ実行により、プログラム実行の流れをビジュアルに観察

演習

資料：**32 ～ 38**

【トピックス】

- **メソッド呼び出し**における**ジャンプ**
- **メソッド**内で使用される変数が**消える**タイミング

Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する（あとで使うので消さないこと）

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.println(foo(100));
        System.out.println(foo(150));
        System.out.println(foo(400));
    }
}
```

Print output (drag lower rig

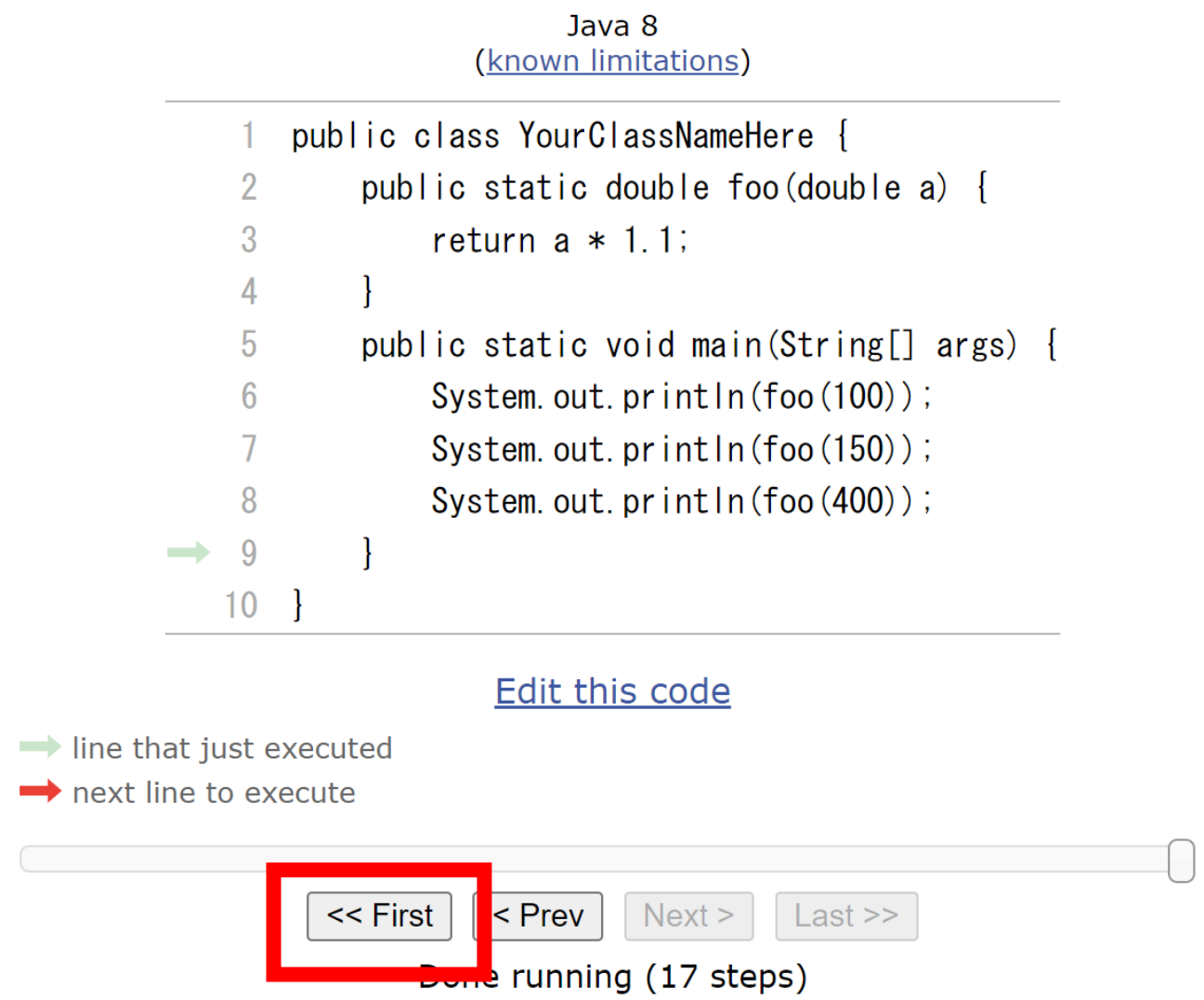
```
110.00000000000001
165.0
440.00000000000006
```

実行結果を確認

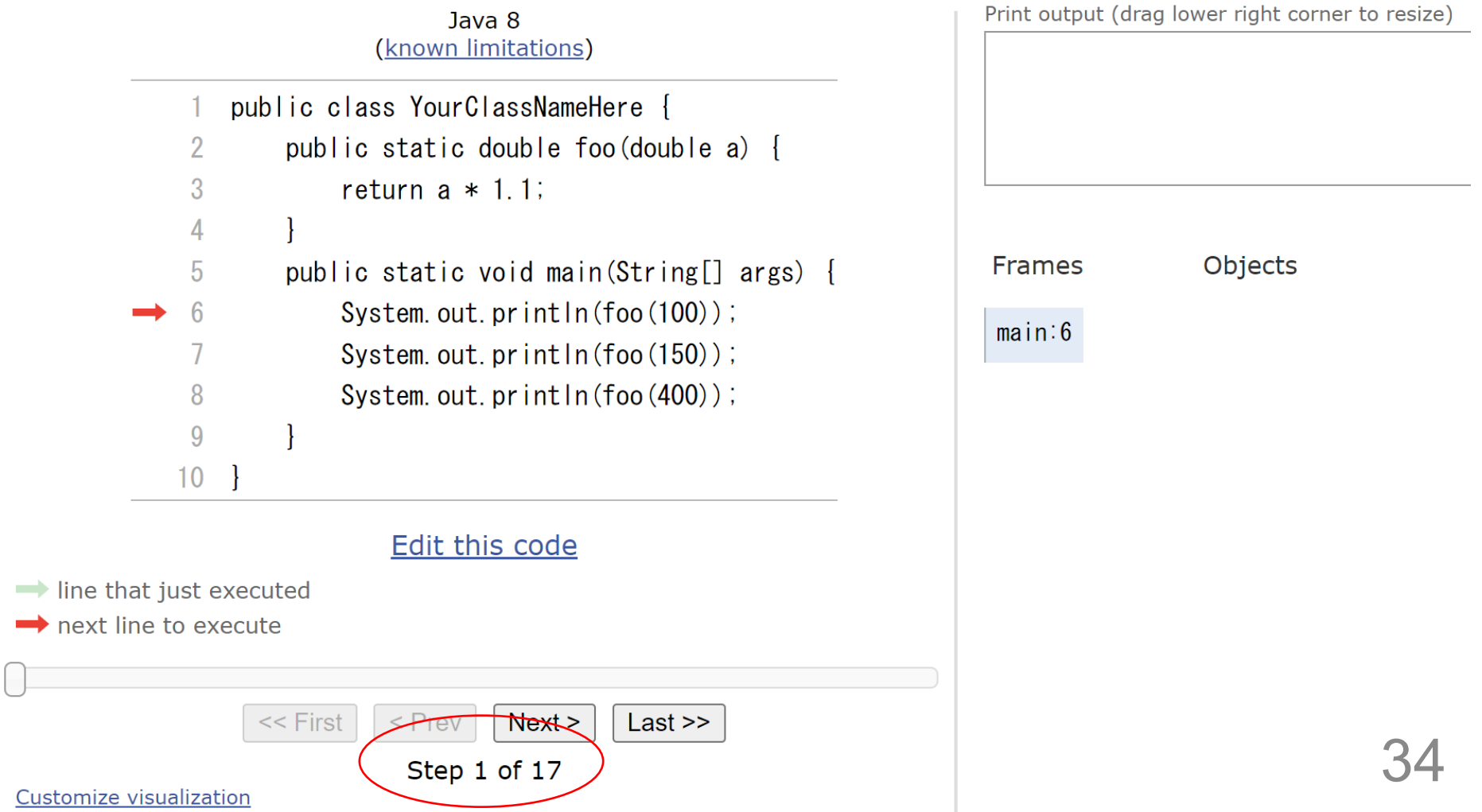
Visualize Execution をクリック，Last をクリック

※ 編集画面に戻るには Edit this code をクリック

②「First」をクリックして，最初に戻る

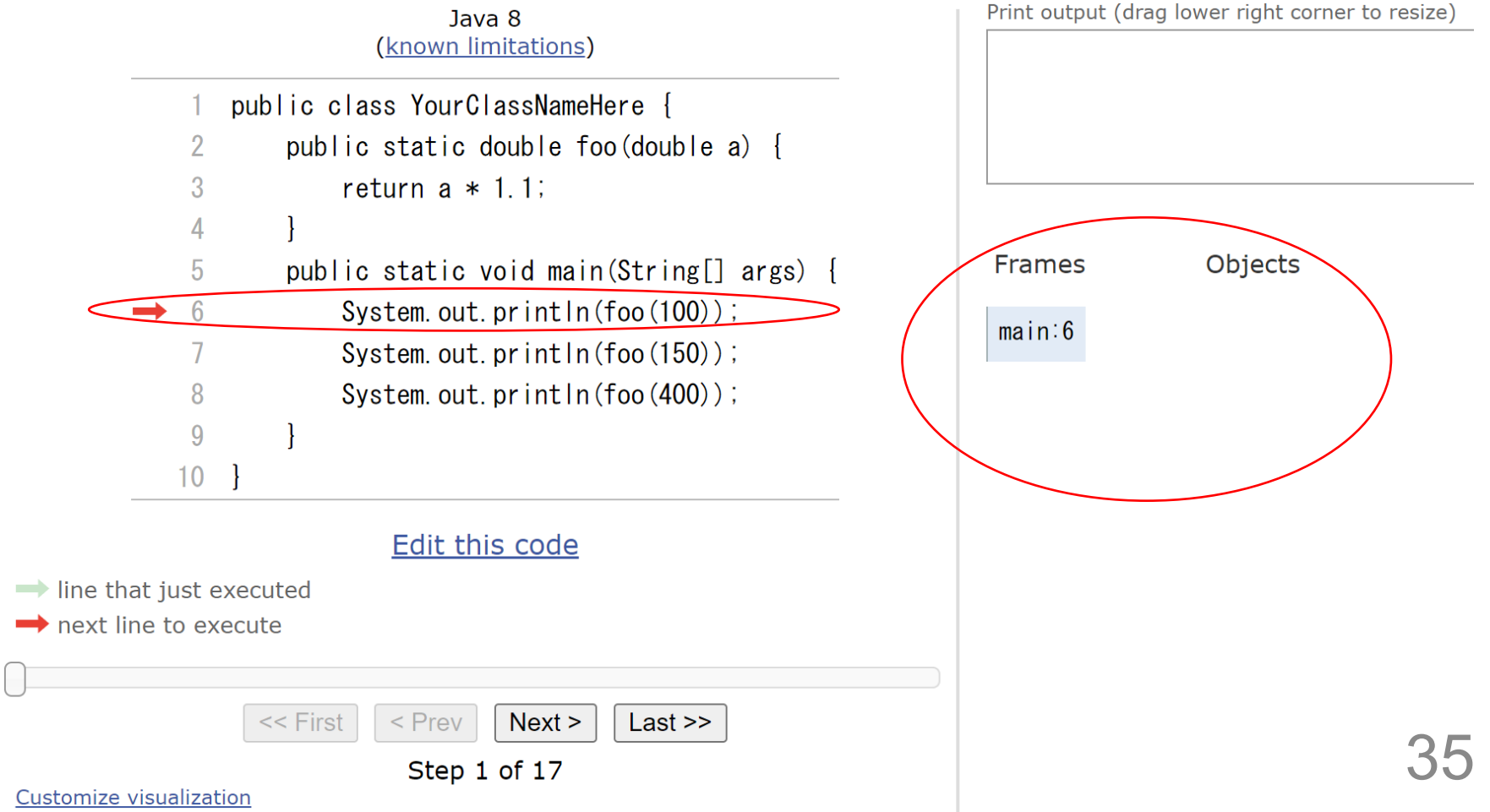


③「**Step 1 of 17**」と表示されているので，全部で，ステップ数は 17 あることが分かる（ステップ数と，プログラムの行数は違うもの）



④ 最初に戻したので

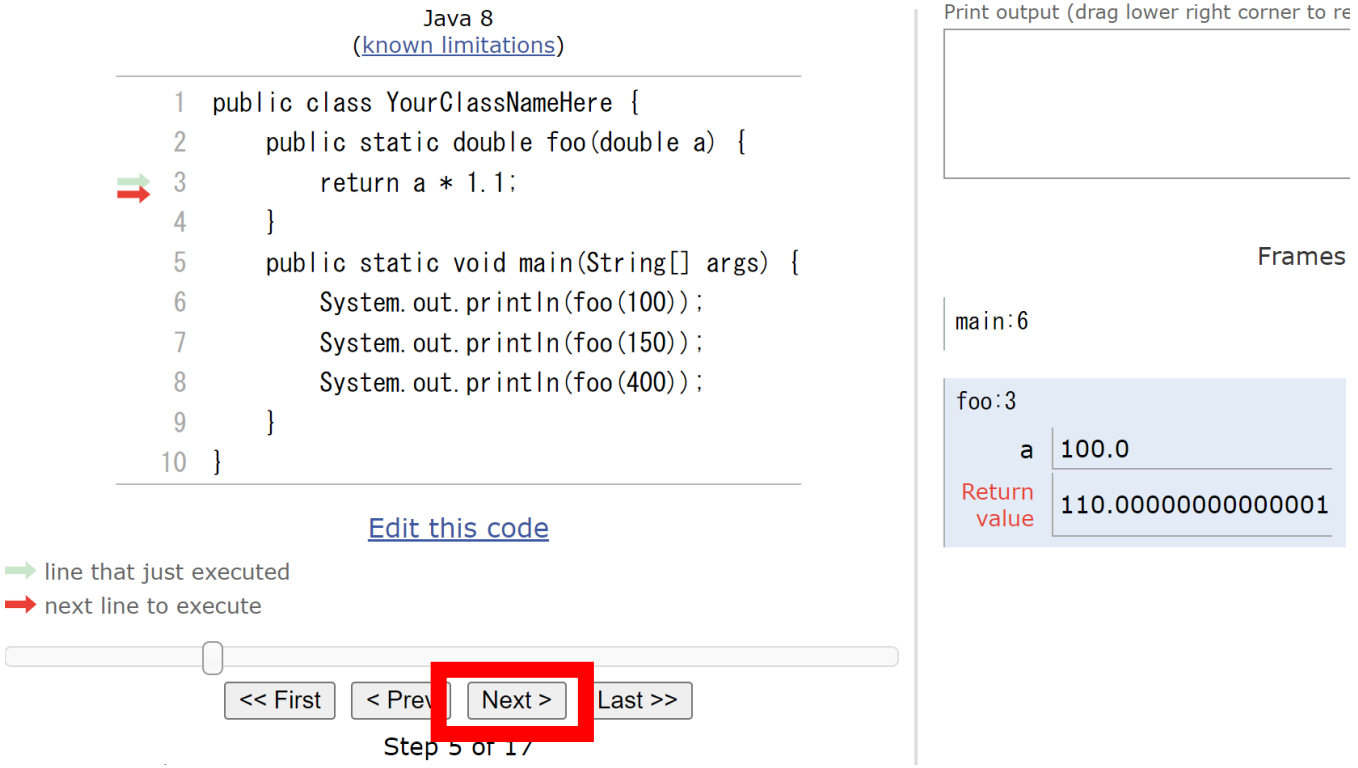
- すべての**オブジェクト**は消えている
- **赤い矢印**は，**main メソッドの先頭**のところに**戻っている**



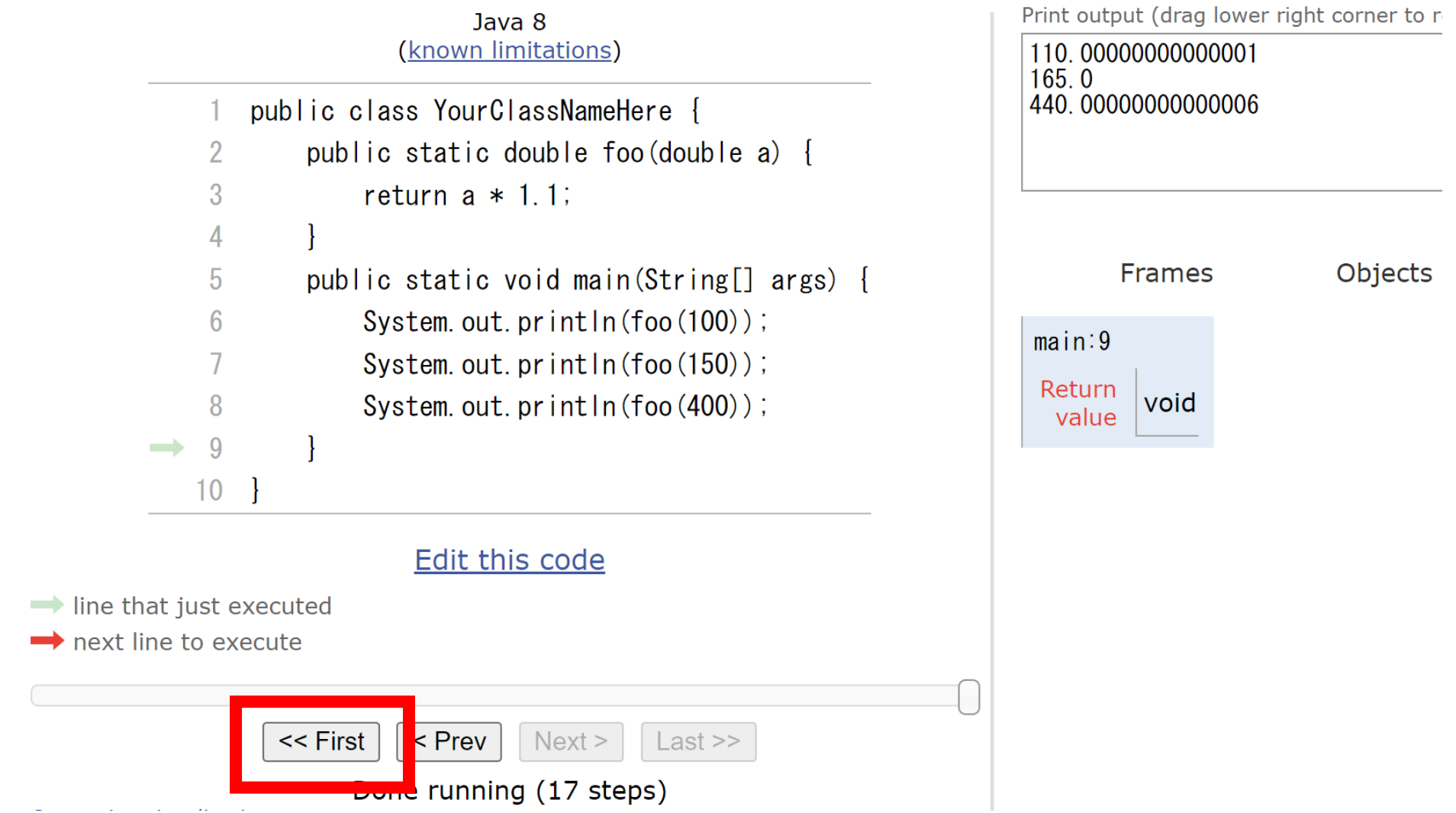
⑤ **ステップ実行**したいので，「**Next**」をクリックしながら，**矢印の動きを確認.**

※「**Next**」ボタンを何度か押し，それ以上進めなくなったら終了

見どころ
main と foo の間で
ジャンプする**ところ**



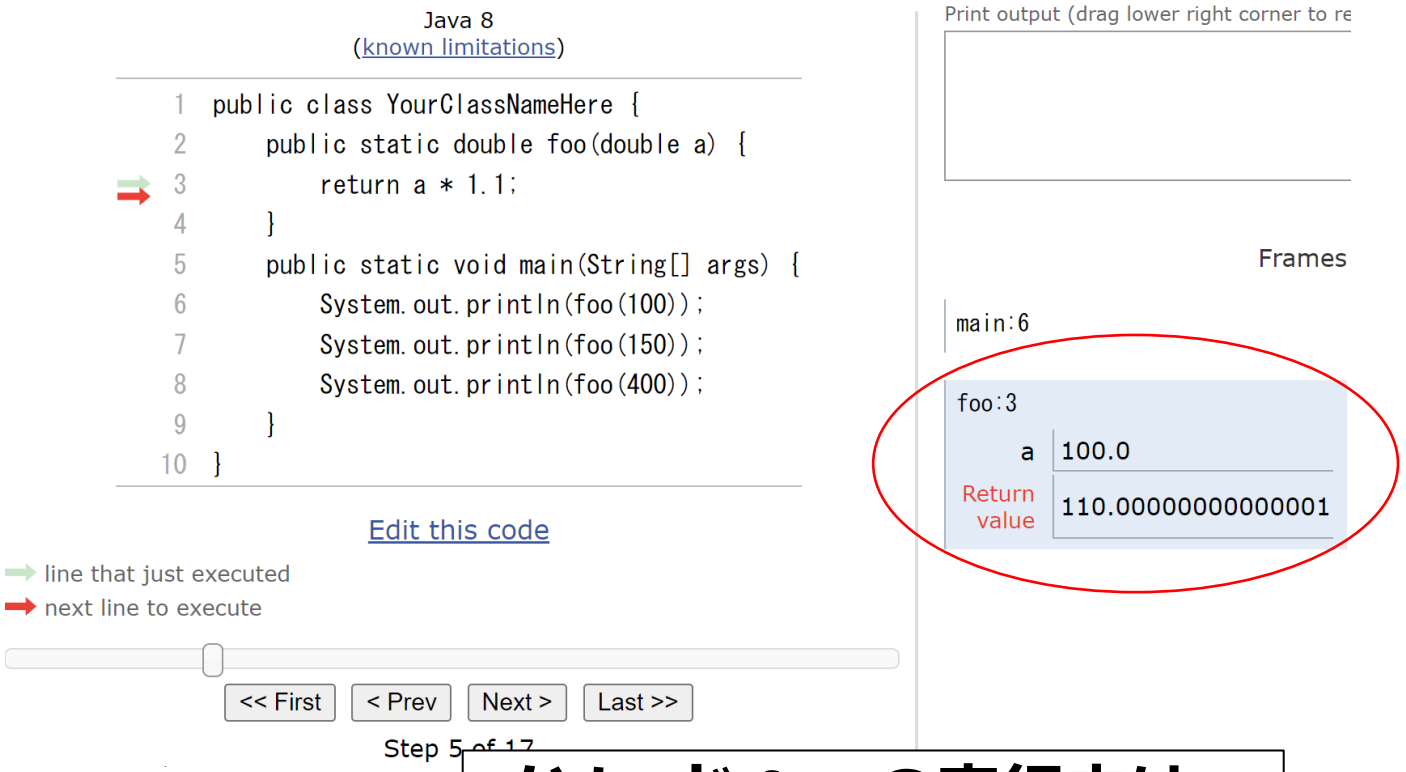
⑥ 終わったら，もう一度，「First」をクリックして，最初に戻る



⑦ もう一度，**ステップ実行.**

今度は、**緑の矢印**を見ながら、**変数 a** が**生成**、**消去**されるタイミングを確認

緑の矢印：いま実行が終わった行



メソッド foo の実行中は，変数 a が現れる

全体まとめ

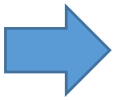
- **抽象化**は ：**変数**を使って，**複数の式**を**１つにまとめる**ことなど

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

- **メソッド呼び出し**：ジャンプ，変数の生成と消去が自動で行われる

関連ページ

- **Java プログラミング入門**

GDB online を使用

https://www.kkaneko.jp/pro/ji/index.html

- **Java の基本**

Java Tutor, GDB online を使用

https://www.kkaneko.jp/pro/pi/index.html

- **Java プログラム例**

https://www.kkaneko.jp/pro/java/index.html

ソースコード 3-2, 3-3

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.println(foo(100));
        System.out.println(foo(150));
        System.out.println(foo(400));
    }
}
```