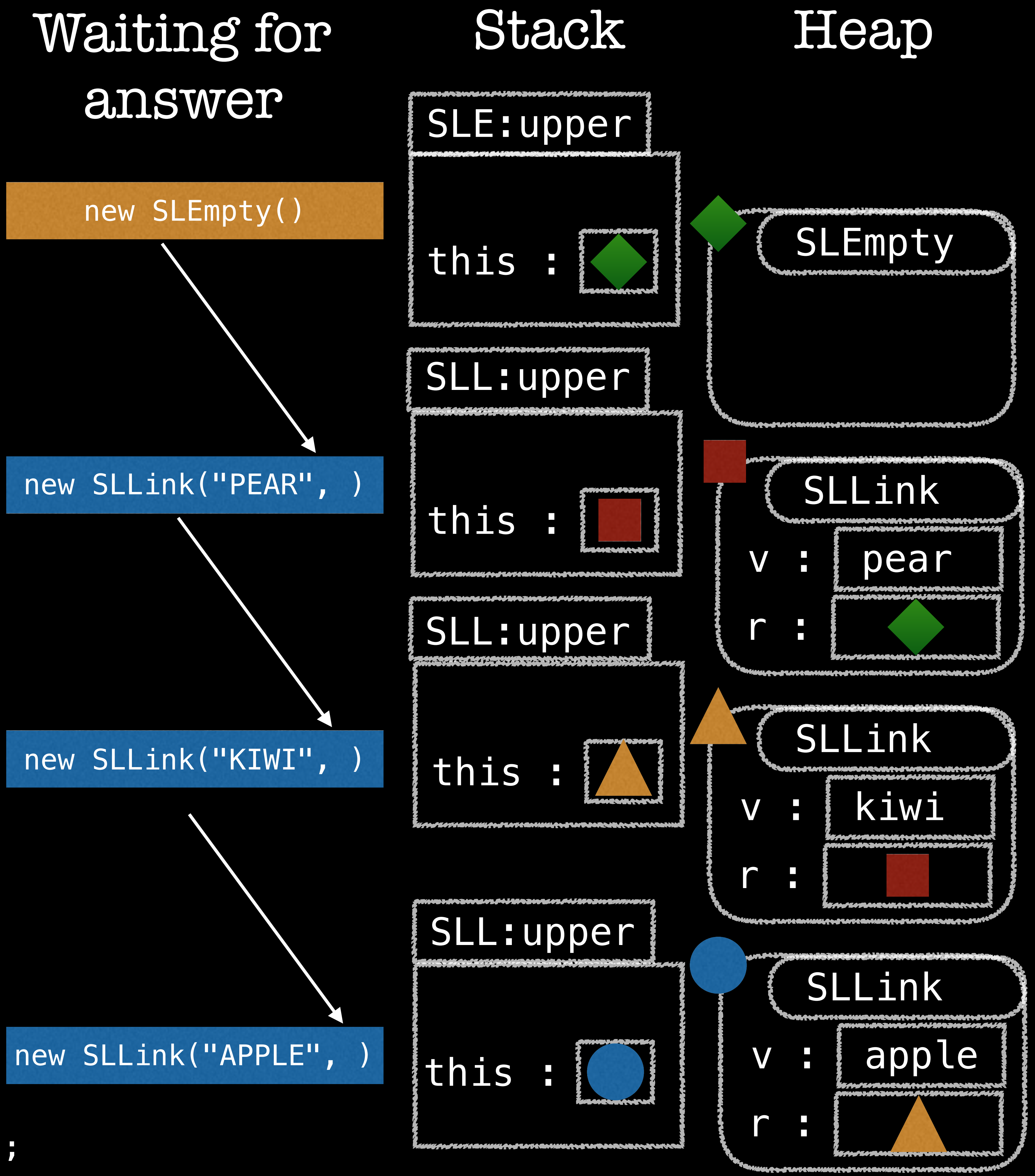
```
interface StringList {
 StringList upper();
}
class SLLink
  implements StringList {
 String value; StringList rest;
 SLLink(
    String value,
    StringList rest) {
  this.value = value;
  this.rest = rest;
 }
 public StringList upper() {
  return new SLLink(
   this.value.toUpperCase(),
   this.rest.upper());
 }
}
class SLEmpty
  implements StringList {
 SLEmpty() { }
 public StringList upper() {
  return new SLEmpty();
 }
}
class ExamplesLink {
 StringList sl =
  new SLLink("apple",
   new SLLink("kiwi",
    new SLLink("pear",
     new SLEmpty()))));
 StringList slu = this.sl.upper();
}
```



Recursive data:
A class where one of the **fields** refers to the class itself, one of its superclasses, or an interface it implements.

```
class SLLink implements StringList {
 String value;
 StringList rest;
 public StringList upper() {
  return new SLLink(
   this.value.toUpperCase(),
   this.rest.upper());
 }
}
class SLEmpty implements StringList {
 …
 public StringList upper() {
  return new SLEmpty();
 }
}
```

Recursive method call:
A call from the body of a method to itself, with new arguments. In our case, the new argument is often just a new value for `this`.

Base case:
A condition or implementation of a method that ends a chain of recursive calls.

Often, recursive data is a strong hint that a recursive method is the right implementation.

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

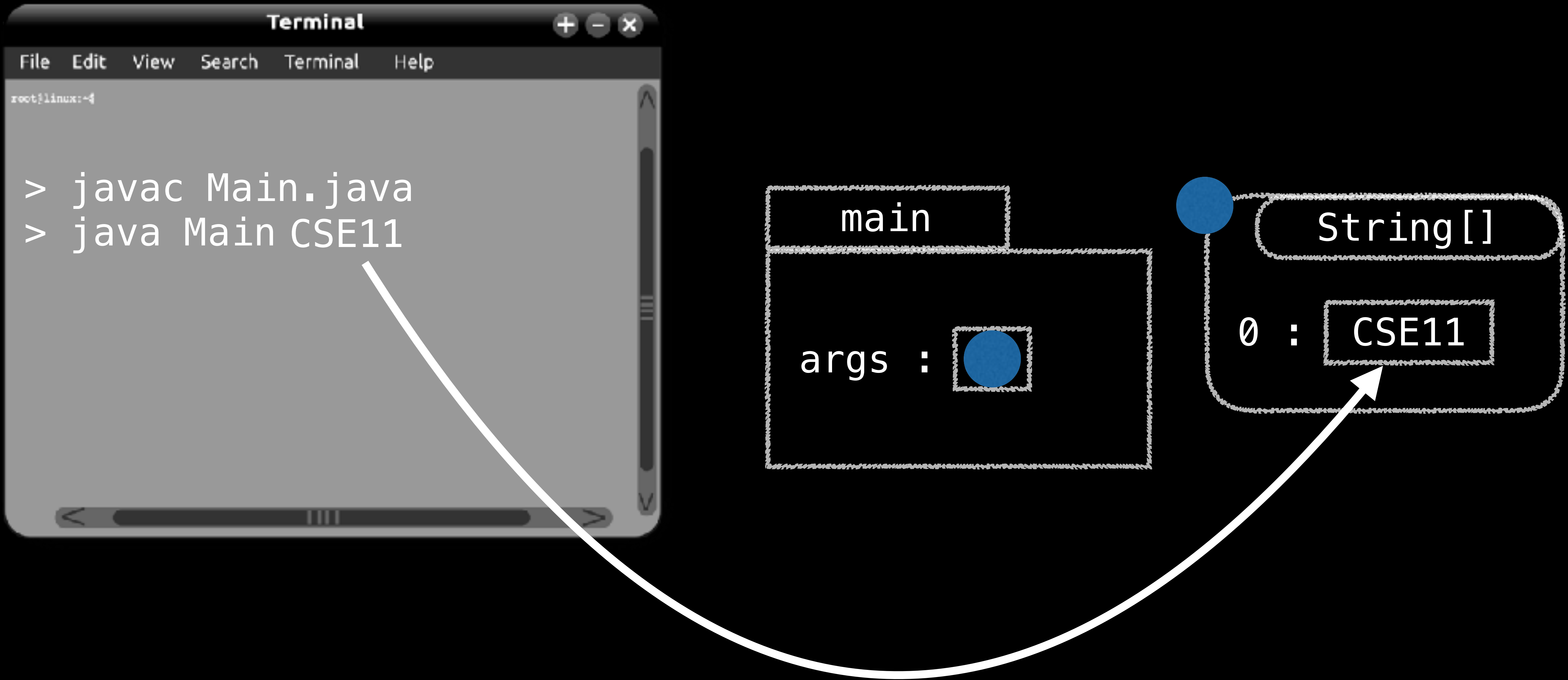
`void` This method has no return value

`static` `main` Java runs the method with the name `main` with the `static` keyword in the given class

`String[] args`
`args[0]` A value that holds the Strings listed on the command line in indices. Called an **Array**.

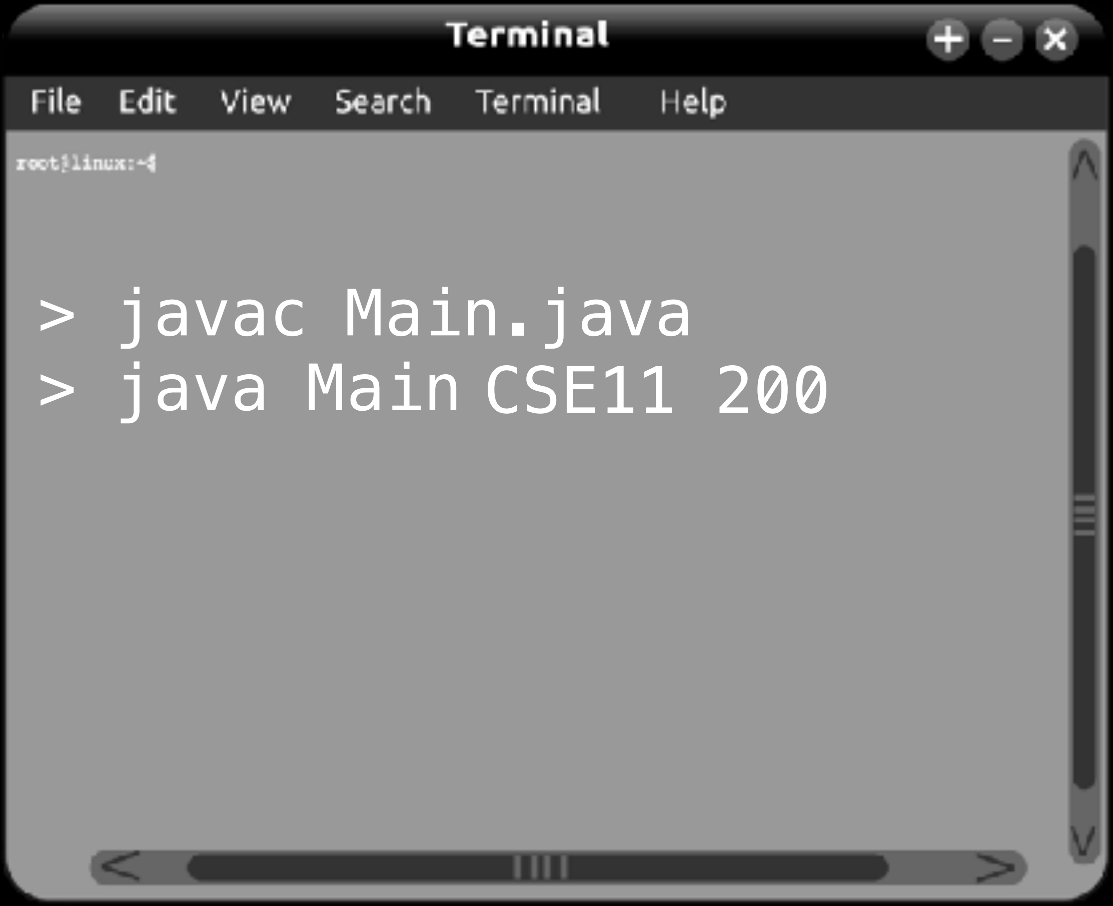
`System.out.println` A method that prints text to the screen

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



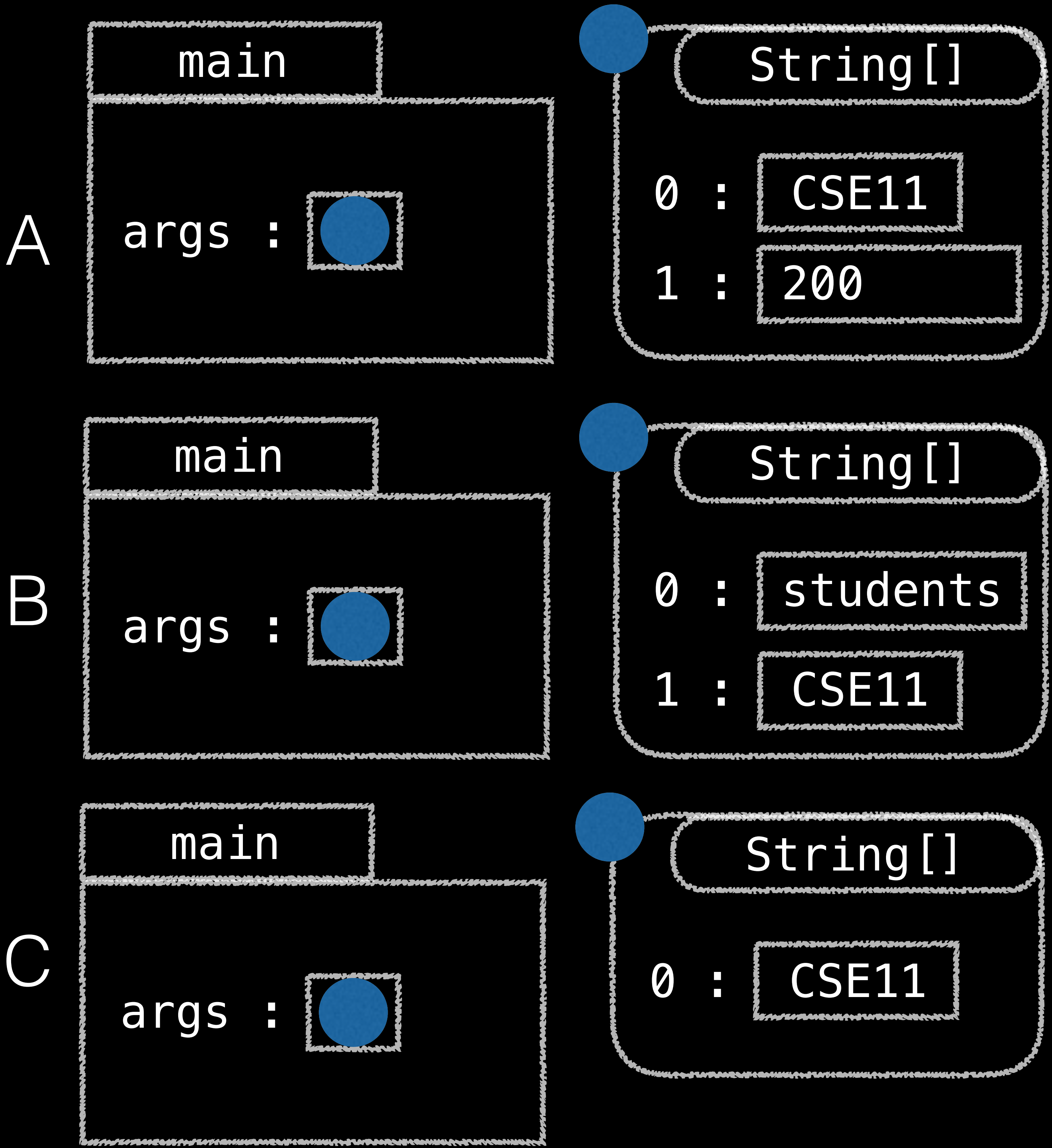
```
> javac Main.java
> java Main CSE11
```

```
class Main {
  public static void main(String[] args) {
    System.out.println(args[0] + " has " + args[1] + " students");
  }
}
```



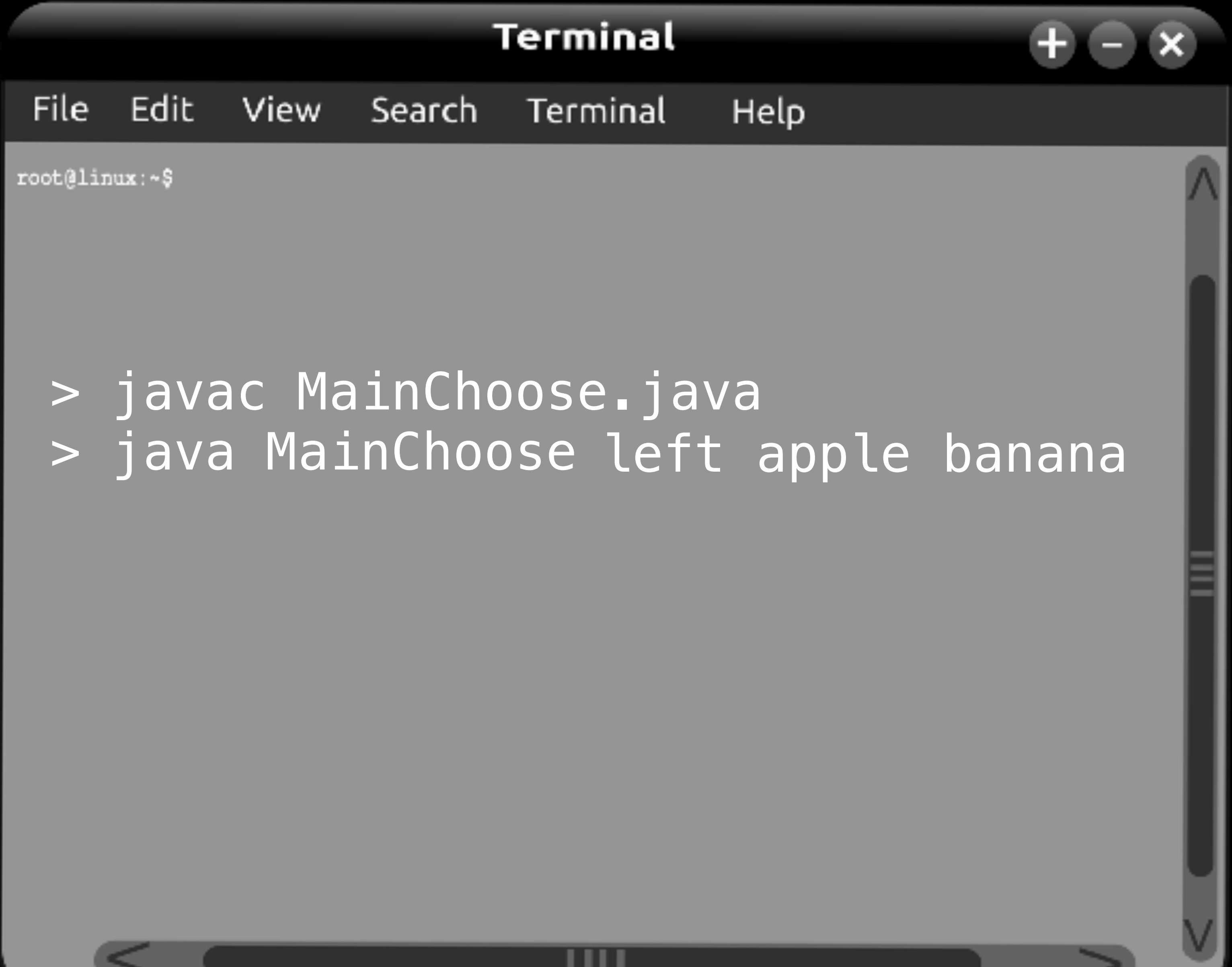
```
root@linux:~$

> javac Main.java
> java Main CSE11 200
```



Which memory diagram matches the progam?

```
class MainChoose {
  public static void main(String[] args) {
    if(args[0].equals("left")) {
      System.out.println(args[1]);
    }
    else {
      System.out.println(args[2]);
    }
  }
}
```



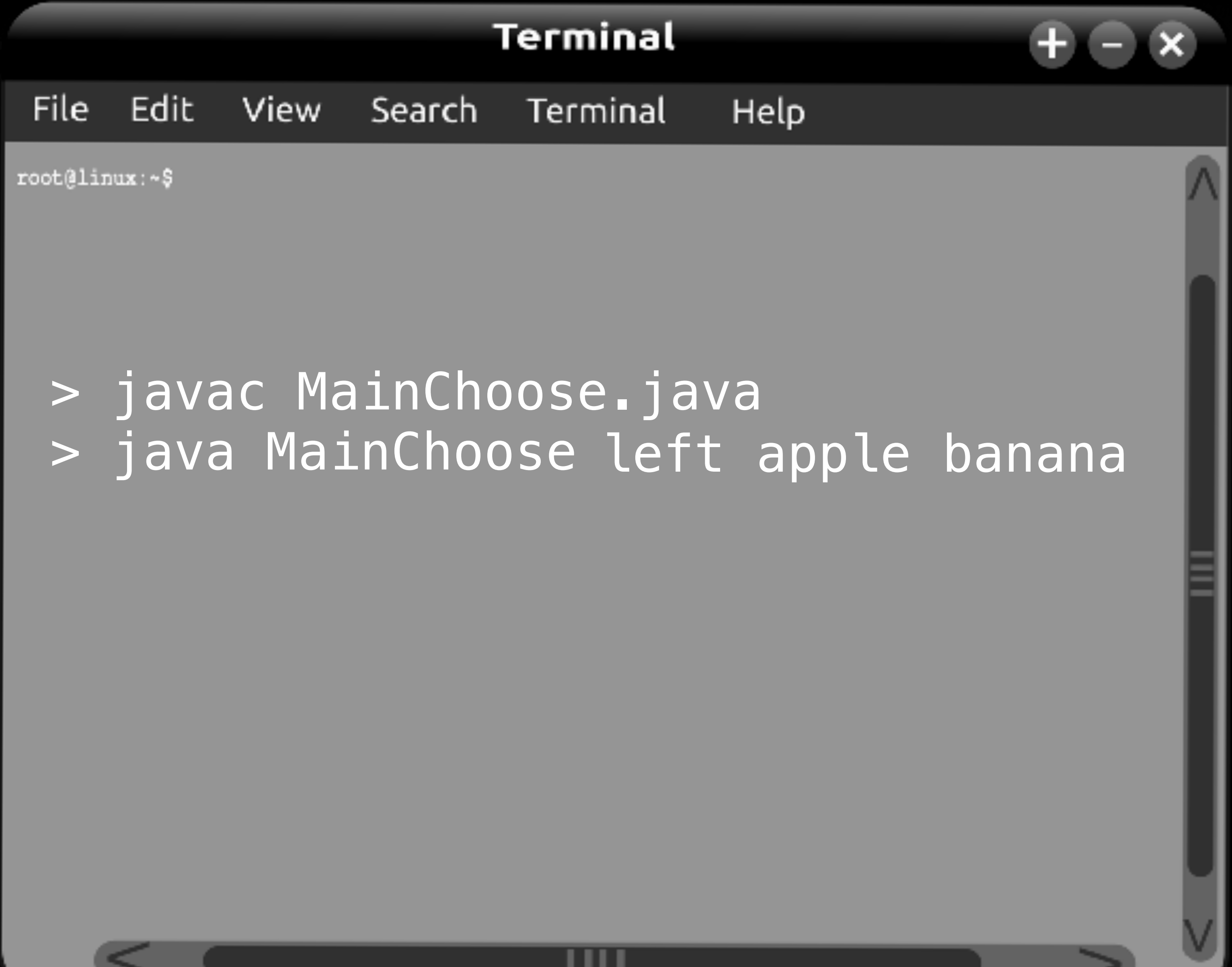
```
root@linux:~$

 > javac MainChoose.java
 > java MainChoose left apple banana
```

What does this print?

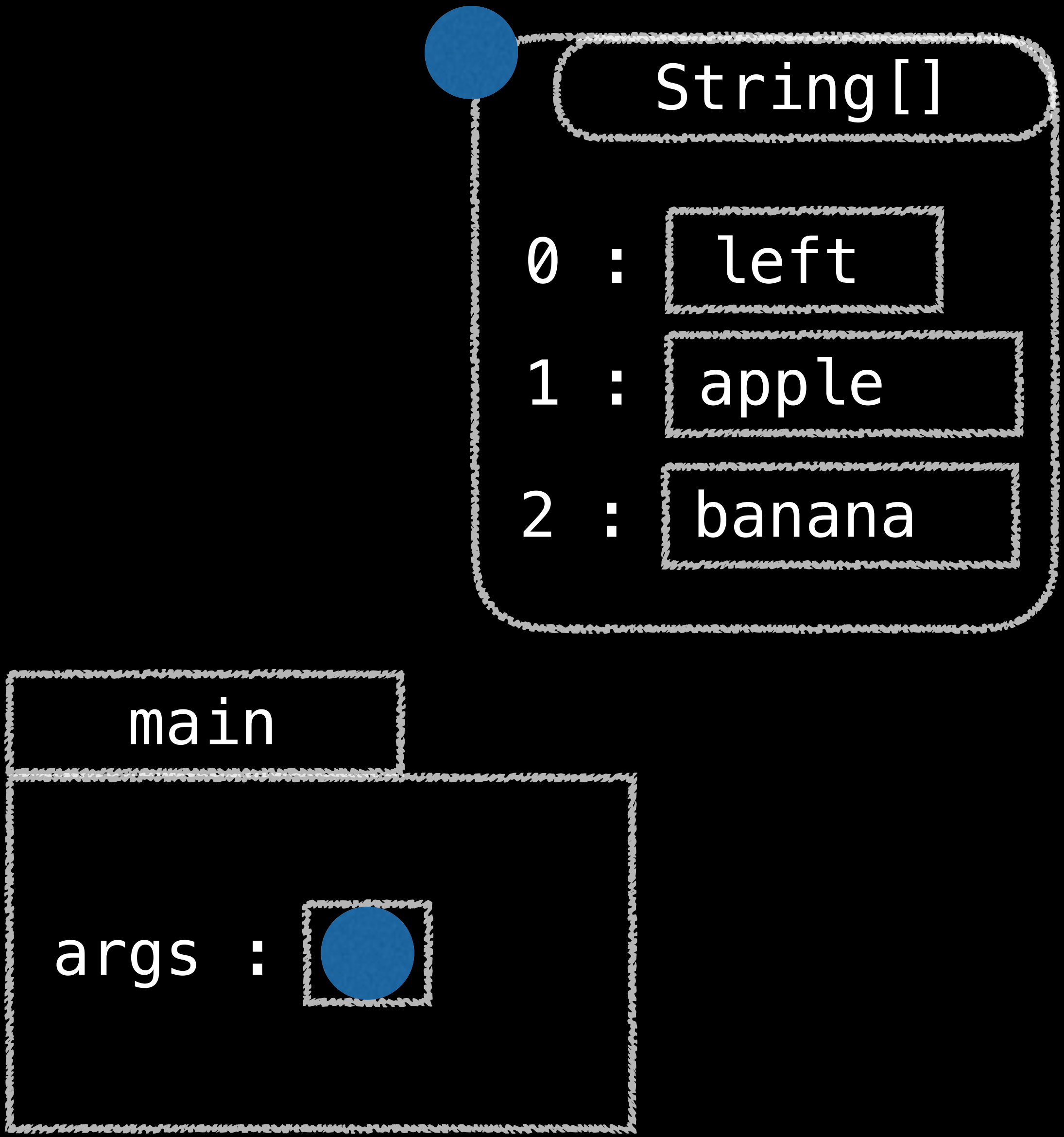
A: apple
B: banana
C: left
D: something else

```
class MainChoose {
  public static void main(String[] args) {
    if(args[0].equals("left")) {
      System.out.println(args[1]);
    }
    else {
      System.out.println(args[2]);
    }
  }
}
```



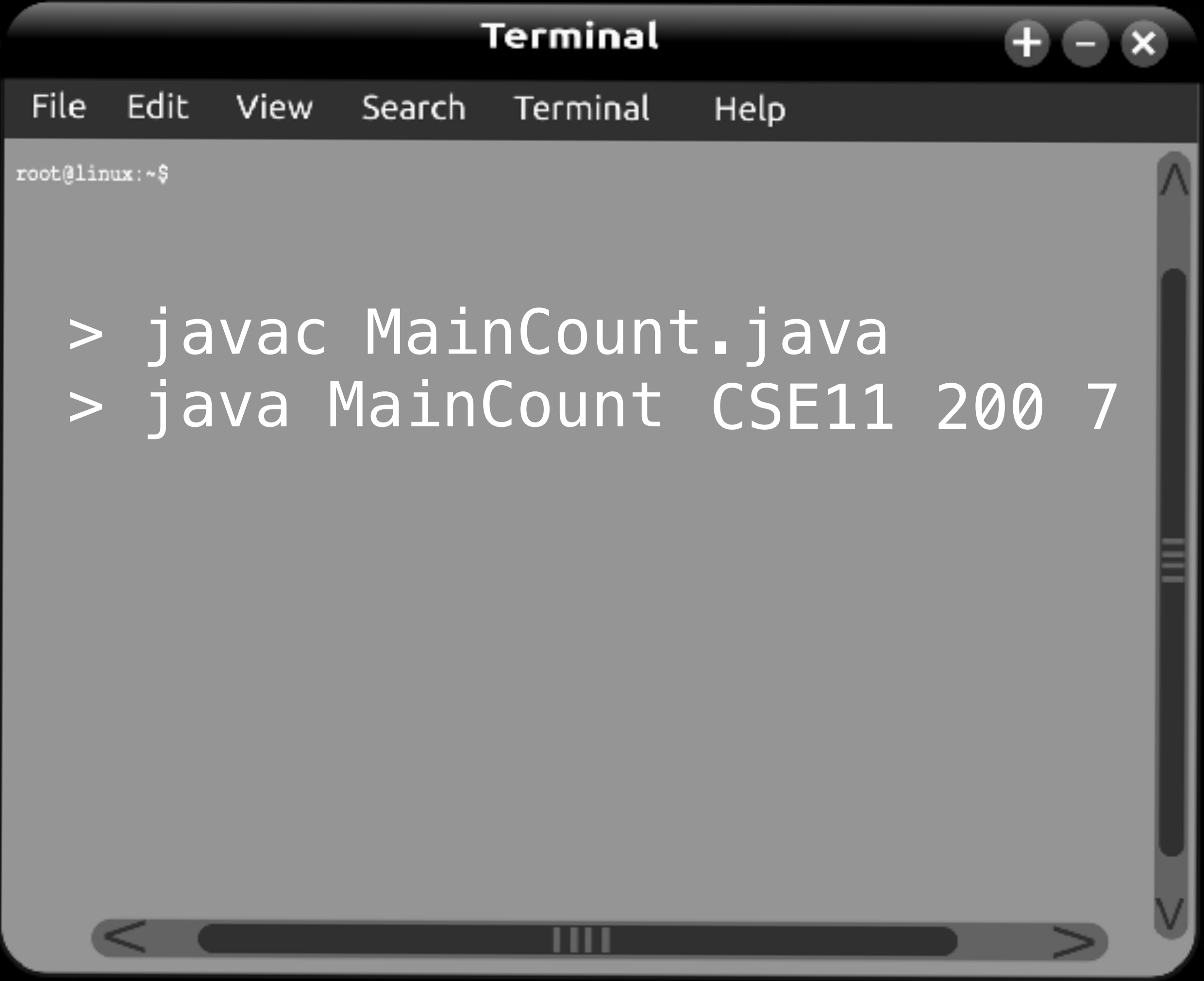
```
root@linux:~$

 > javac MainChoose.java
 > java MainChoose left apple banana
```



```
class MainCount {
  public static void main(String[] args) {
    int students = args[1];
    int staff = args[2];
    int ratio = students / staff;
    System.out.println(
      "There are " + ratio + " students/staff in " + args[0]
    );
  }
}
```

Error: cannot convert String to int

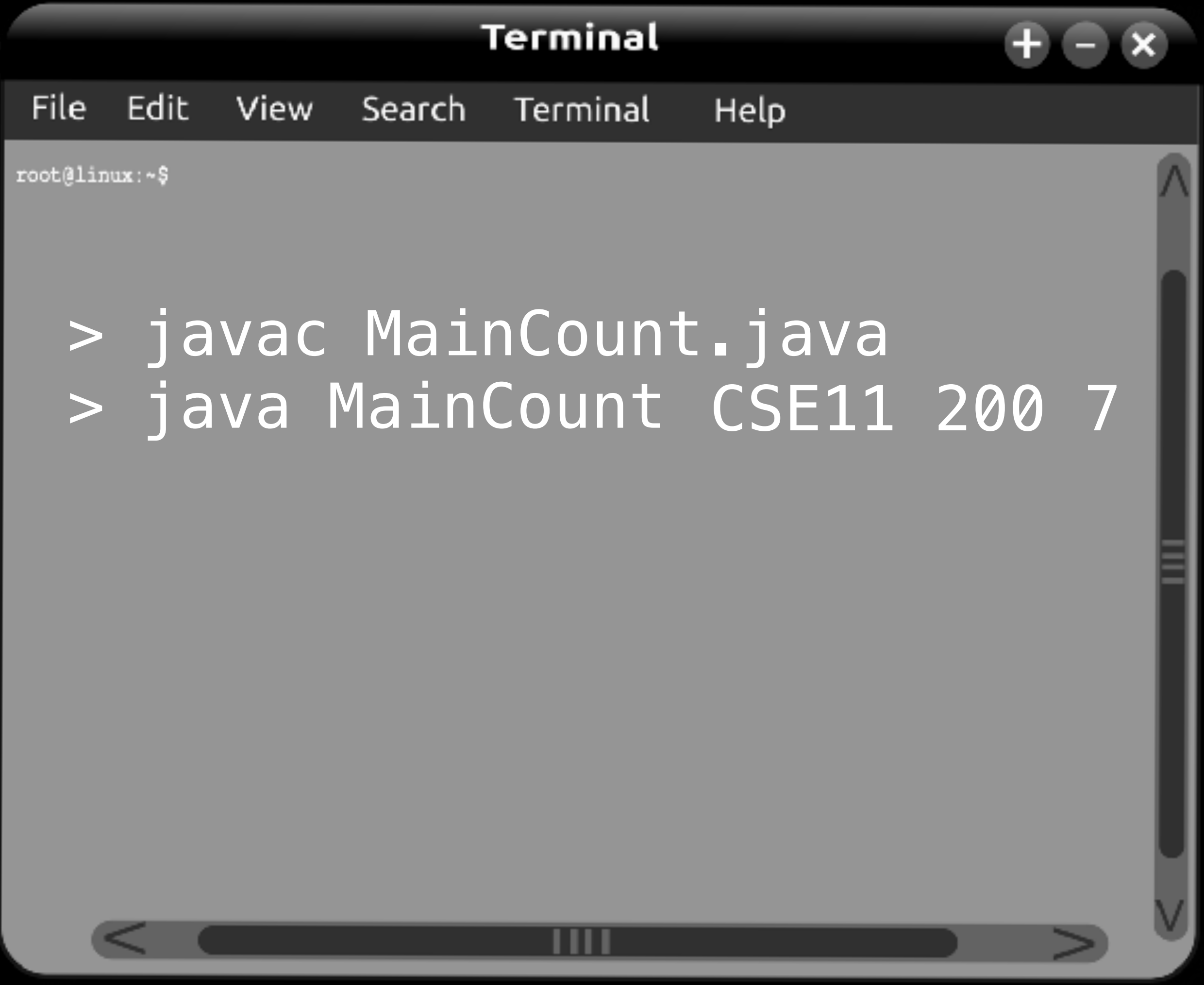


```
> javac MainCount.java
> java MainCount CSE11 200 7
```

What does this print?

A: "There are 28 … in CSE11"
B: "There are CSE11 … in 28"
C: Something else
D: Nothing, it's an error

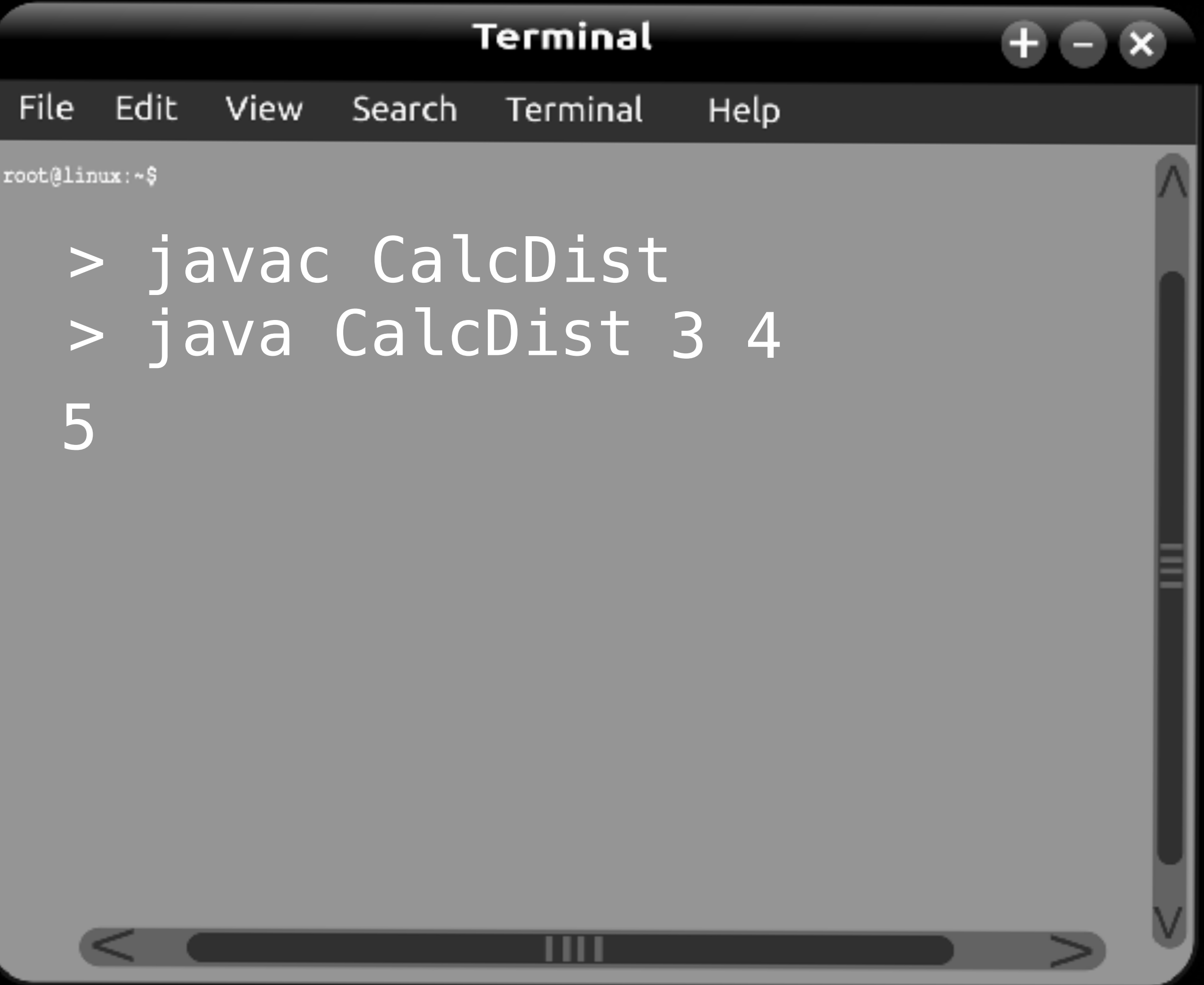
```
class MainCount {
  public static void main(String[] args) {
    int students = Integer.parseInt(args[1]);
    int staff = Integer.parseInt(args[2]);
    int ratio = students / staff;
    System.out.println(
      "There are " + ratio + " students/staff in " + args[0]
    );
  }
}
```



```
> javac MainCount.java
> java MainCount CSE11 200 7
```

Parse: to turn a string into another kind of data (more useful for this application)

```java
class Point {
  double x, y;
  Point(double x, double y) {  this.x = x; this.y = y; }
  double distToOrigin() {
    return Math.sqrt(Math.pow(this.x, 2), Math.pow(this.y, 2));
  }
}
class CalcDist {
  public static void main(String[] args) {
    // DO NOW!  fill this in
  }
}
```

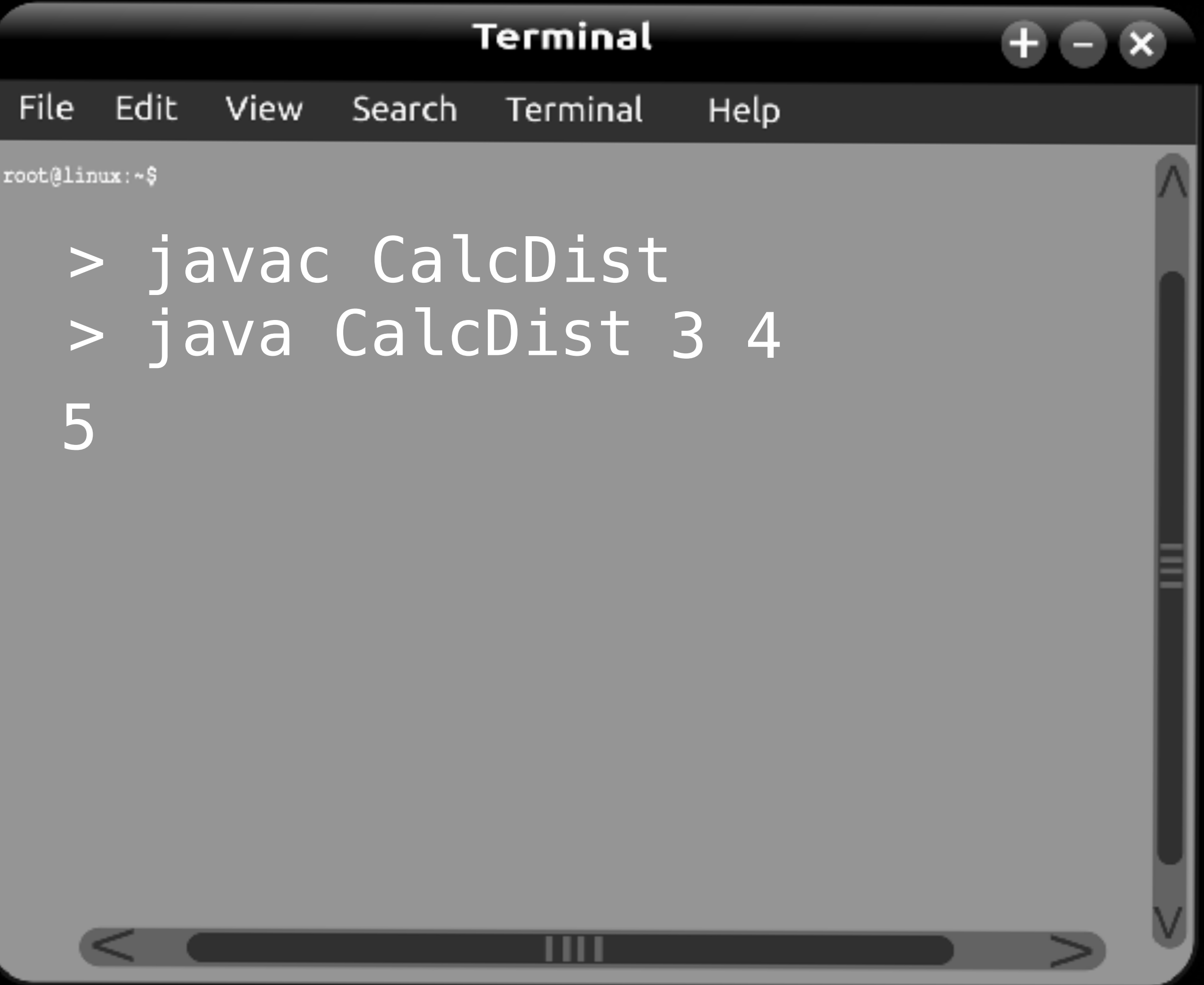


```
root@linux:~$
 > javac CalcDist
 > java CalcDist 3 4
5
```

Write a class called CalcDist that takes 2 command-line arguments, treats them as x and y coordinates, and calculates the distance from 0 using the Point class.

`Double.parseDouble(args[n])`

```
class Point {
  double x, y;
  Point(double x, double y) {  this.x = x; this.y = y; }
  double distToOrigin() {
    return Math.sqrt(Math.pow(this.x, 2), Math.pow(this.y, 2));
  }
}
class CalcDist {
  public static void main(String[] args) {
    Point p = new Point(
      Double.parseDouble(args[0]),
      Double.parseDouble(args[1]));
    System.out.println(p.distToOrigin());
  } }
```



```
root@linux:~$

 > javac CalcDist
 > java CalcDist 3 4
 5
```

Write a class called CalcDist that takes 2 command-line arguments, treats them as x and y coordinates, and calculates the distance from O using the Point class.

`Double.parseDouble(args[n])`