Recap: Java & OOP Concepts

CSC207 Fall 2015

Computer Science
UNIVERSITY OF TORONTO

stack
1. stores info about active subroutine of the program
2. to keep track of the point to which each active subroutine should return control when it finishes executing.

Memory

Data is stored in two areas of memory:

Stack (also known as the *call stack):*

- Keeps track of method calls, including parameter values, local variables, and return addresses.

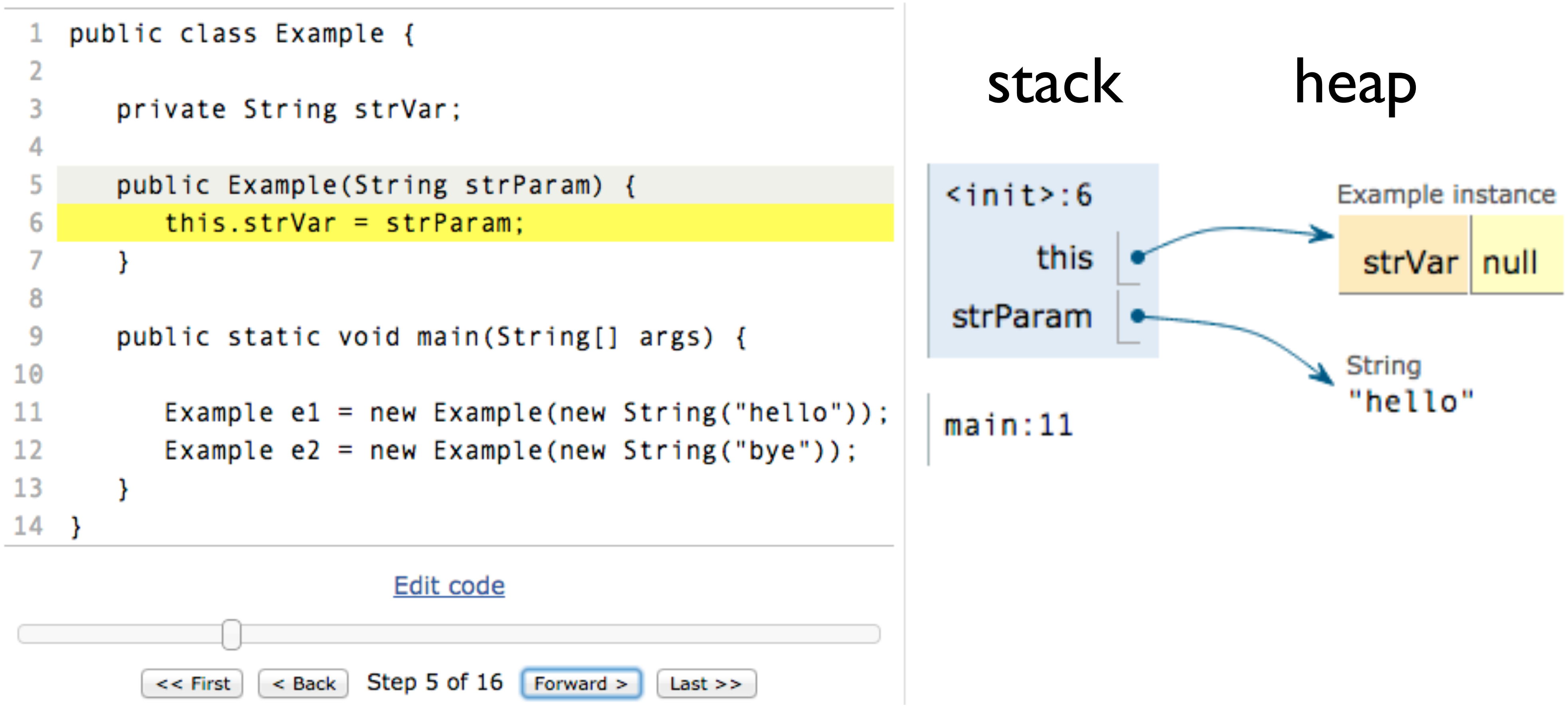
Heap: memory space

- Stores objects.
- Variables on the call stack reference the objects stored in the heap.
- Objects in the heap also reference other objects in the heap.

Garbage collection is a process which removes dead objects from Java Heap space and returns memory back to Heap in Java.

Memory: example (1)

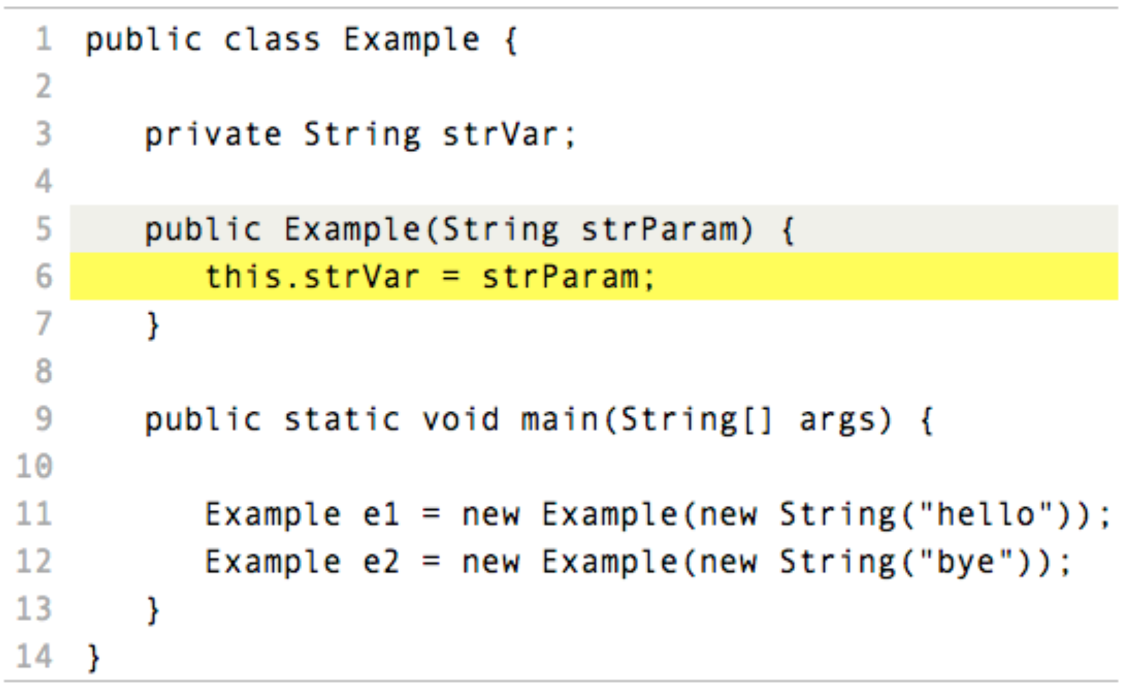
During the first call to method Example.



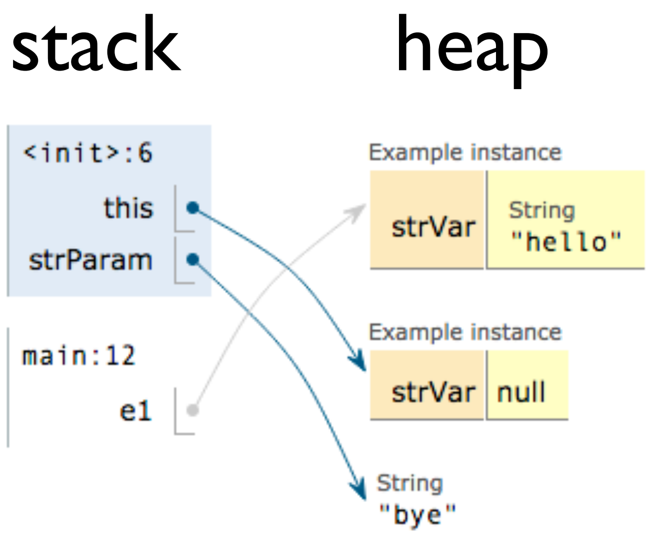
```
public class Example {

   private String strVar;

   public Example(String strParam) {
      this.strVar = strParam;
   }

   public static void main(String[] args) {

      Example e1 = new Example(new String("hello"));
      Example e2 = new Example(new String("bye"));
   }
}
```

Memory: example (2)

During the second call to method Example.

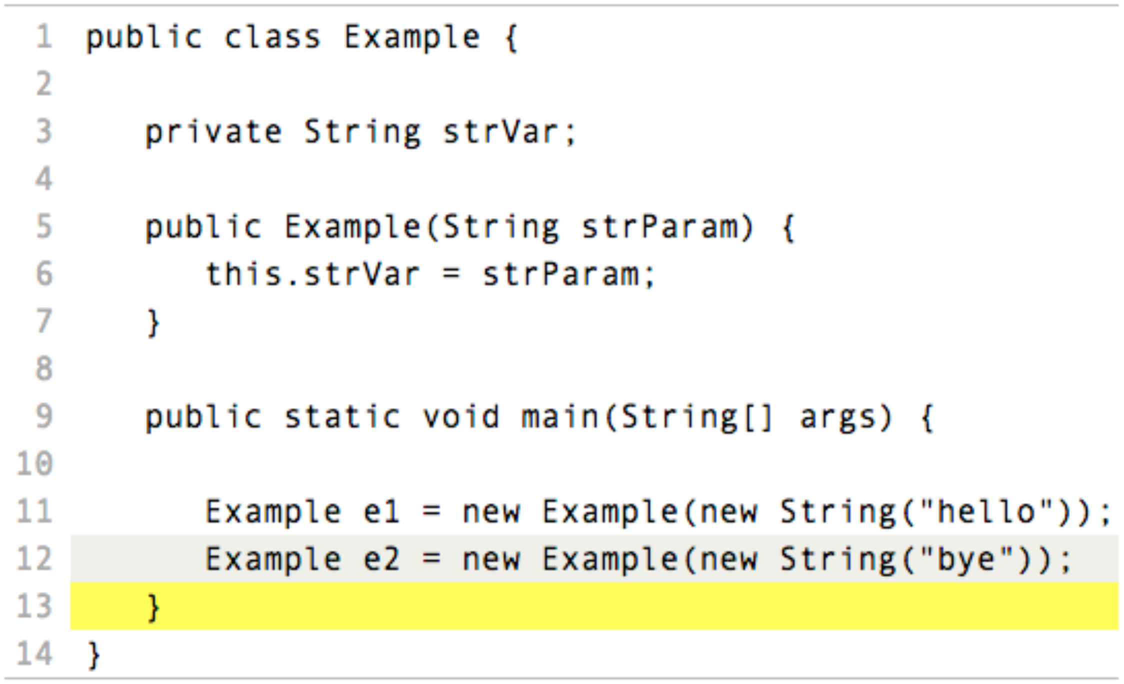


```
 1  public class Example {
 2
 3      private String strVar;
 4
 5      public Example(String strParam) {
 6          this.strVar = strParam;
 7      }
 8
 9      public static void main(String[] args) {
10
11          Example e1 = new Example(new String("hello"));
12          Example e2 = new Example(new String("bye"));
13      }
14  }
```

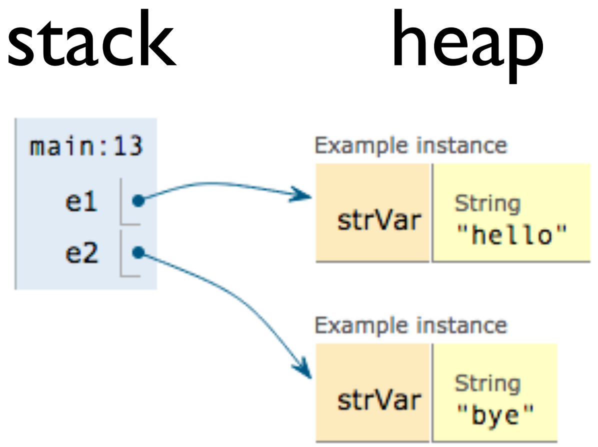


Memory: example (3)

After both calls to method Example are complete.



```
 1  public class Example {
 2
 3     private String strVar;
 4
 5     public Example(String strParam) {
 6        this.strVar = strParam;
 7     }
 8
 9     public static void main(String[] args) {
10
11        Example e1 = new Example(new String("hello"));
12        Example e2 = new Example(new String("bye"));
13     }
14  }
```



Garbage collection

What happens to objects that are no longer in use?

An used object is also called an unreferenced object, because it is no longer referenced by any part of the program.

In Java, unused objects are automatically deallocated by the garbage collector.

Details about when and how this is done are beyond the scope of this course.

Other languages, like C, require programmers to manually allocate and deallocate memory.

Recap

Generics

Abstract classes

Interfaces

a type or method to operate on objects of various types while providing compile-time type safety.
making more of your bugs detectable at compile time.

Generics

Generics allow types to be parameters to classes, interfaces, and methods.

Code that uses generics:

- Allows type checking at compile time. The aim is to detect bugs at compile time, rather than at run time.

- Eliminates the need for typecasting.

have to think about it later

Non-generic code vs. Generic Code

Non-generic code:

```
List firstList = new ArrayList();
firstList.add("Hello");
firstList.add(new Integer(3));
```

since it adds object by default?

```
// To call a String method, we would need to typecast:
((String) firstList.get(0)).charAt(0);

// But if we typecast an non-String to type String, a
// runtime exception occurs.
((String) firstList.get(1)).charAt(0);
```

Generic code:

```
List<String> secondList = new ArrayList<>();
secondList.add(“Hello”);
secondList.add(new Integer(3)); // compile error
```

Creating generic array

of any type T

```java
public class Example<T> {

   private T[] myArray;

   public Example(int size) {

      // This results in a compilation error:
      //myArray = new T[size];

      // Here is an alternate approach that you may use.
      myArray = (T[]) new Object[size];

      // In CSC324, you'll likely learn about better approaches.
   }
}
```

Our WaitList Example

Non-generic WaitList:

```
public class WaitList {

    // The queue items are Objects.
    private ConcurrentLinkedQueue content;
    …
}
```

Non-generic WaitList:

```
public class WaitList<E> {

    // The queue items have type E.
    private ConcurrentLinkedQueue<E> content;
    …
}
```

Abstract class

May contain instance and static (class) variables.

May contain abstract methods.

May contain implemented methods.

Cannot be instantiated.

A class can *extend* an abstract class.

Interface

May contain only `public static final` variables.

May contain abstract methods.

Typically do not contain implemented methods.

Cannot be instantiated.

what is implemented methods

A class can *implement* one or more interfaces.

Why interfaces?

An interface serves as a formal contract that is checked by the compiler.

When a class states that it implements an interface, the code will not compile unless all methods declared in the interface are implemented in the class.

APIs

The basic idea of interfaces is not Java-specific.

An Application Programming Interface (API) describes software in terms of its operations (functions/methods), types, and inputs/outputs.

The API does not provide implementation details. In fact, implementations can change.

You are currently relying on the Java API!

Examples of other APIs include Dropbox, Twitter, YouTube, C standard template library, Cocoa, Android, etc.