Lecture 12 – Exceptions

08-671
Java for Application Programmers

February 18, 2016

Terry Lee
Assistant Teaching Professor
School of Computer Science

08-671 Lecture Topics

(subject to change – but only a little bit)

#1 Intro

#2 Primitive Types

#3 Java Classes

#4 Reference Types

#5 Loops & Arrays

#6 Methods & Classes

#7 Lists & Maps

#8 File & Network I/O

#9 Swing Interfaces

#10 Swing Actions

#11 Threads

#12 Exceptions

#13 Functional Programming

#14 In-class Written Exam

* Programming Exam – this will be a 3-hour exam

Exam Plan

- Written Exam
  - In-class on Feb 25th (Thursday)
  - Location: BH A51 (Giant Eagle Auditorium)
  - Plan: multiple choice & fill-in the blank, etc.
    - Closed everything. Pencils, erasers and CMU ID

- Programming Exam
  - Date and Time: 5:30pm on Mar 1st (Tuesday)
  - Location: BH A51 (Giant Eagle Auditorium)
  - Plan: same as HW#6, but different
    - Need your laptop. Don't forget your power adapter

Homework #7

- Due at 11:59pm on Feb 22 (Mon)
  - No extensions!
- Read the spec carefully
  - **Especially, with regard to the relationship between threads and buttons**
  - Hint: Not the same as Lights2.java example!

Outline

✓ Course Planning

⇢ Questions

Detailed Picture of Memory

Recursion

Exceptions

More on Classes (`this`, `super` & casting)

More on Threads

Questions

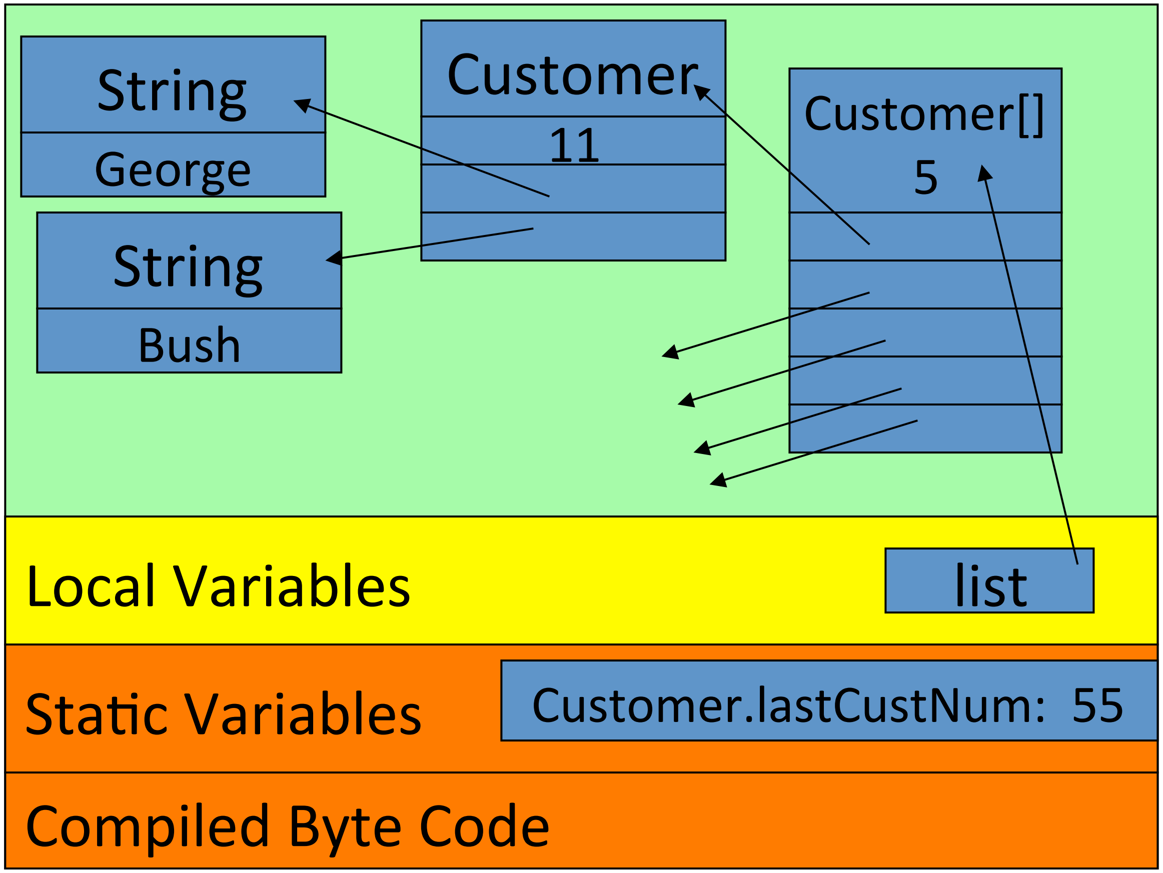
Remember This Example? (Lecture 6)

```
public class Customer {
    private static int lastCustNum = 0;
    public static int getNumCustomers() {
        return lastCustNum / 11;
    }

    private int    customerNumber;
    private String firstName;
    private String lastName;

    public Customer(String first, String last) {…}

    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public int getCustomerNumber(){ return customerNumber; }
    public void setFirstName(String first) { firstName = first; }
    public void setLastName(String last) { lastName = last; }
    public String toString() { … }
}
```

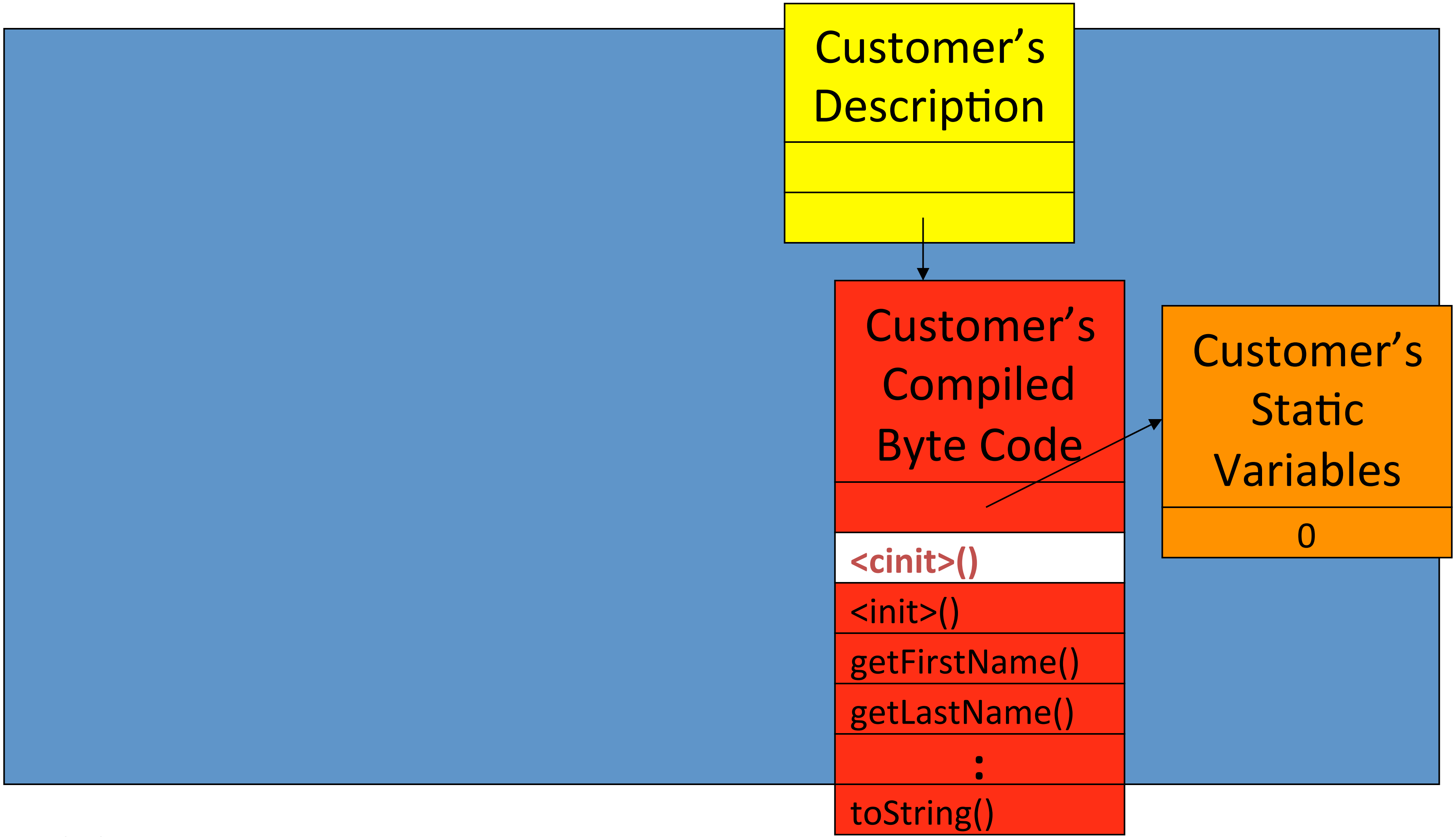
Remember something like this too? (lecture 6)

```
public class CustomerTest {
    public static void main(String[] args) {
        Customer[] list = new Customer[5];
        Customer c = new Customer("Jeb", "Bush");
        list[0] = c;
        list[1] = new Customer("George", "Bush");
    }
}
```

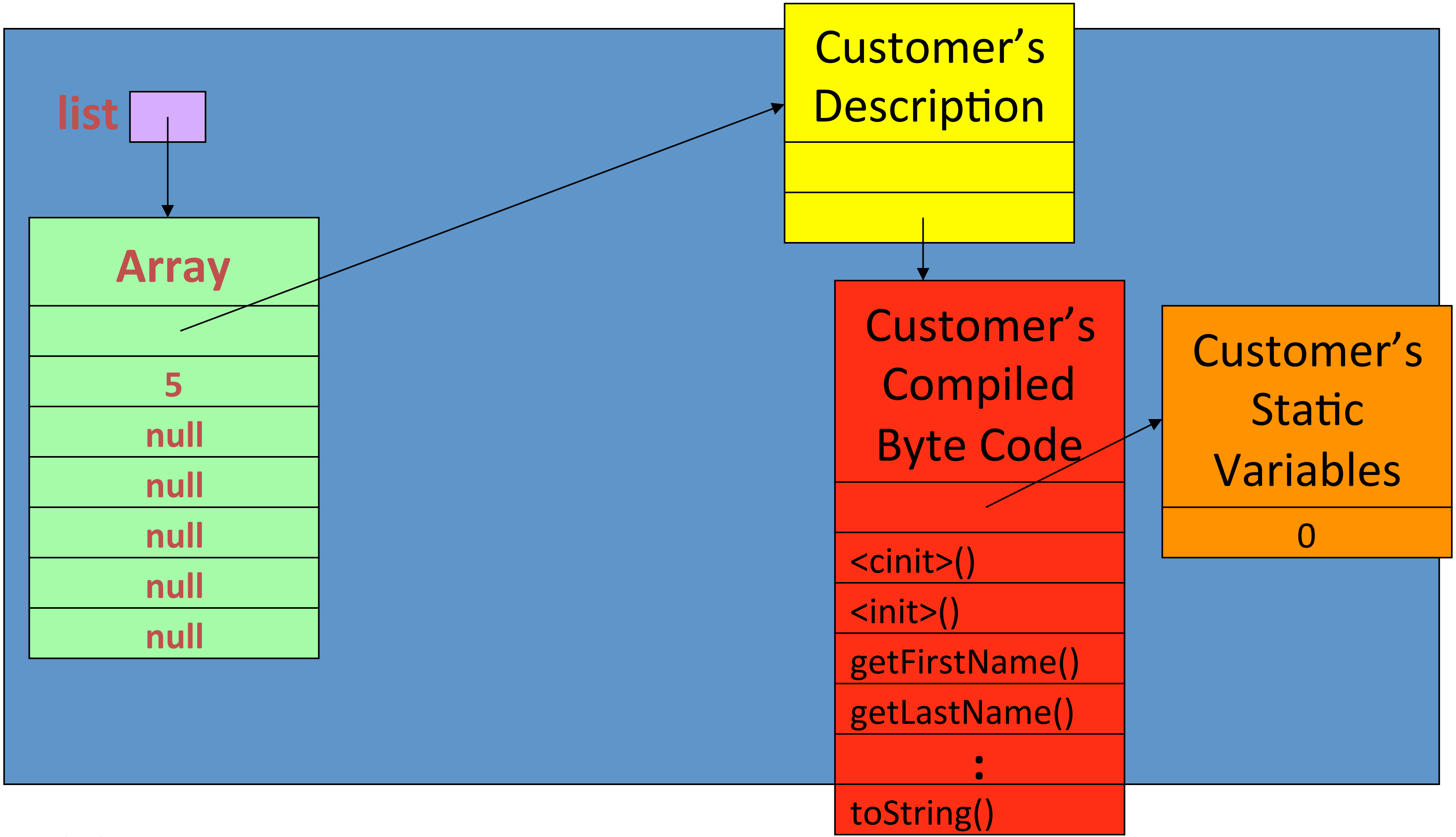
Remember This Picture?



More detailed picture: Loading the Class



```
Customer[] list = new Customer[5];
```



list

Array

5

null

null

null

null

null

Customer’s Description

Customer’s Compiled Byte Code

<cinit>()

<init>()

getFirstName()

getLastName()

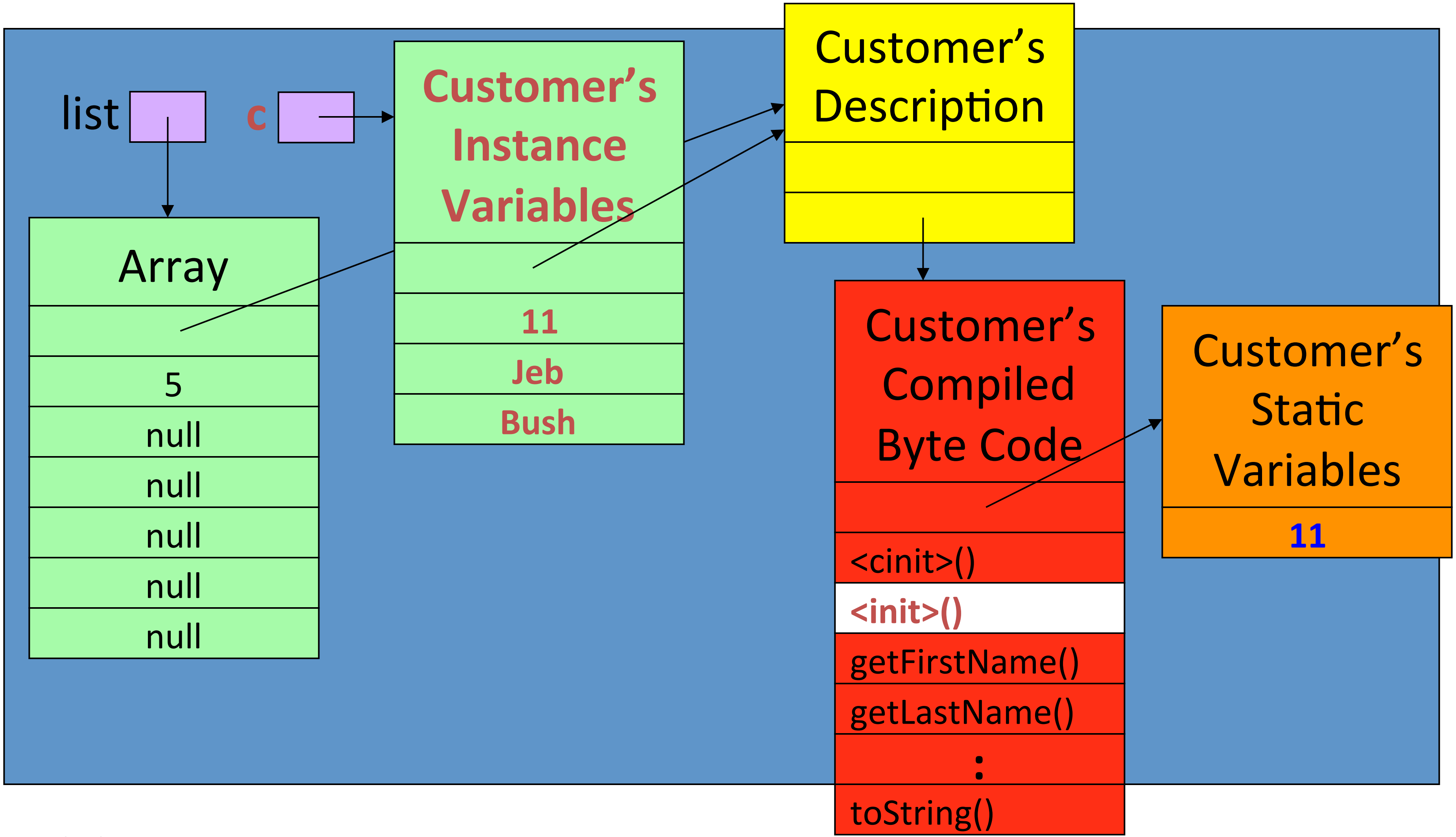
:

toString()

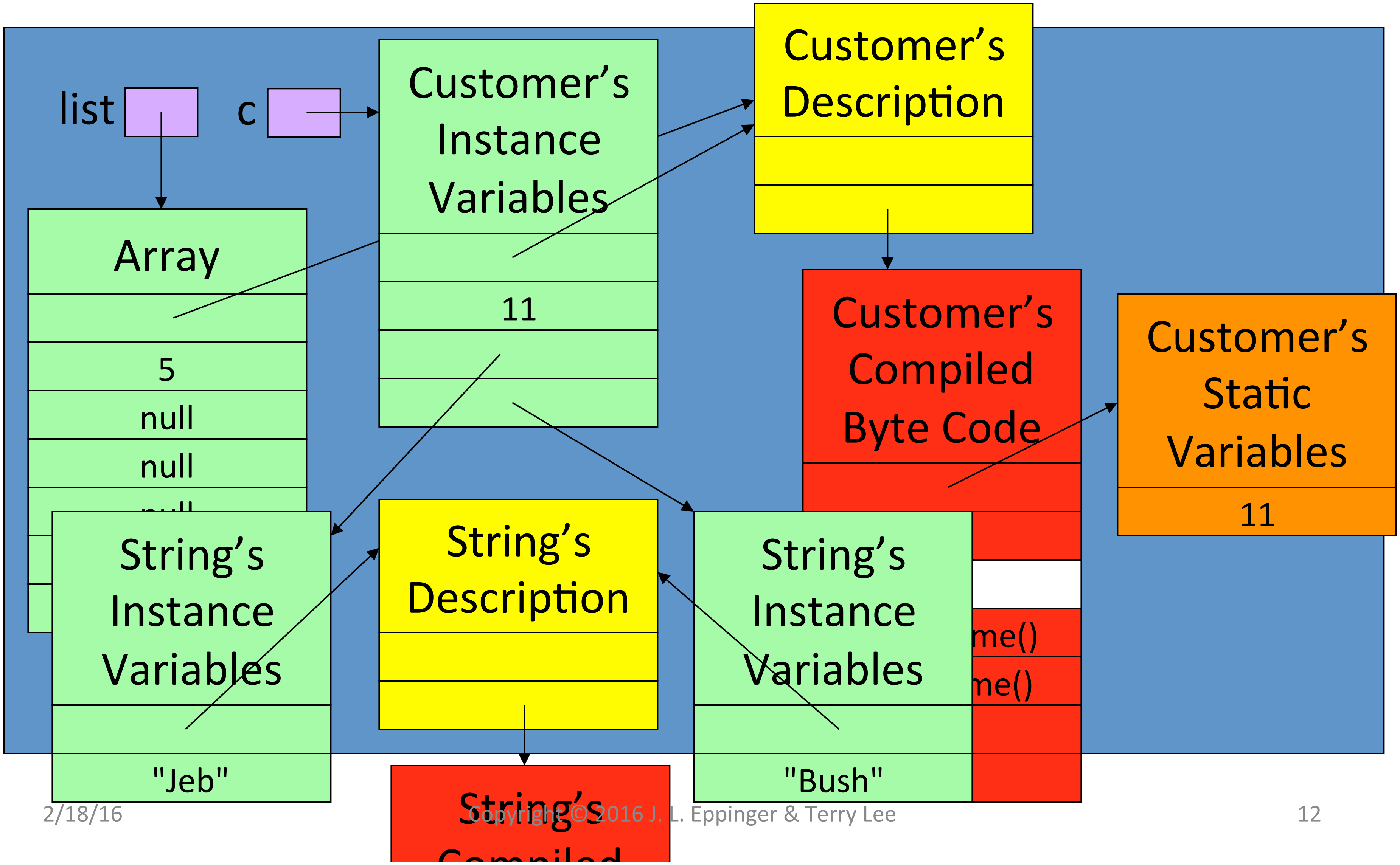
Customer’s Static Variables

0

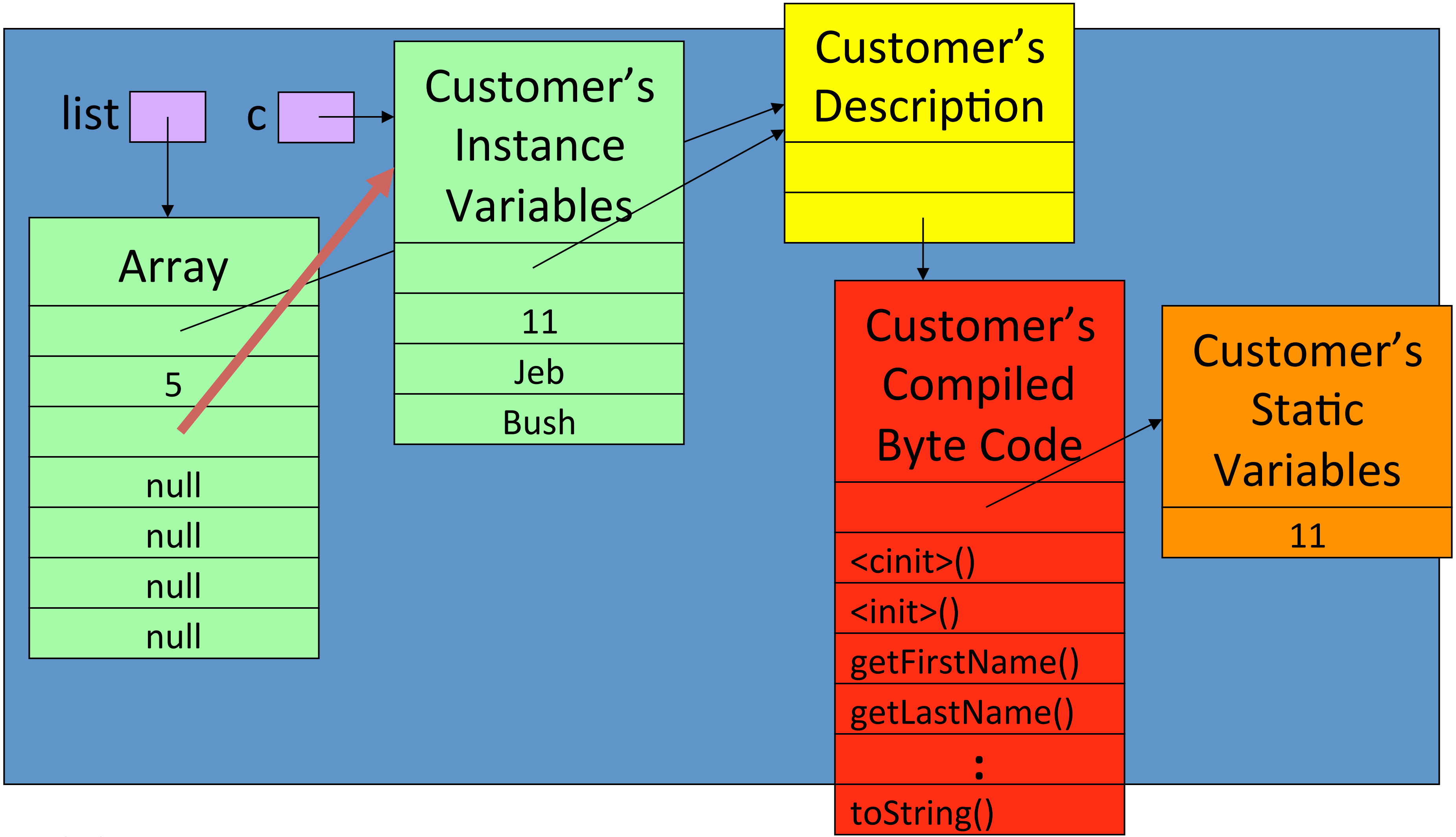
```
Customer c = new Customer("Jeb","Bush");
```



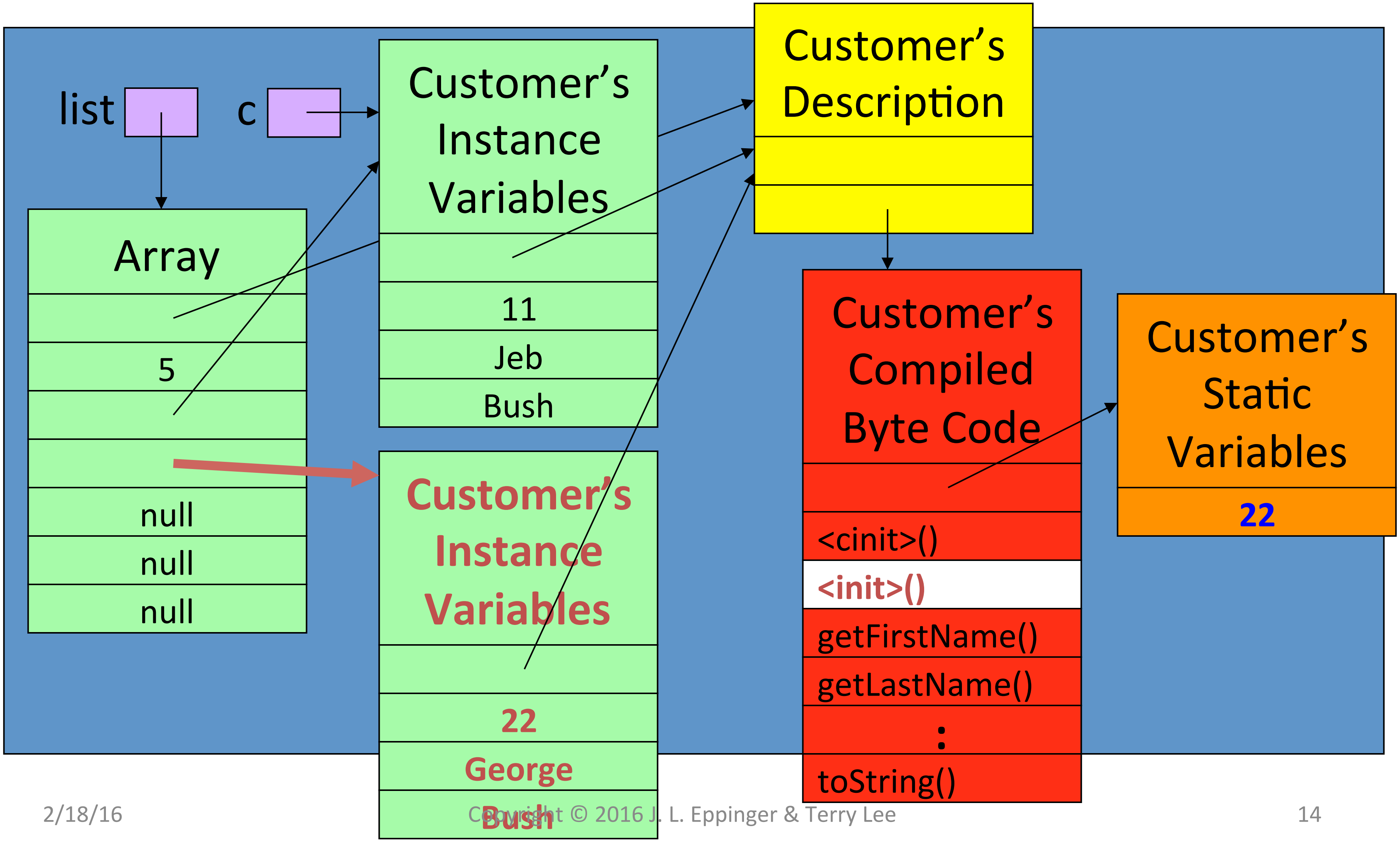
```
Customer c = new Customer("Jeb","Bush");
```


list
c
Customer's Instance Variables
11
Customer's Description
Array
5
null
null
null
Customer's Compiled Byte Code
me()
me()
Customer's Static Variables
11
String's Instance Variables
"Jeb"
String's Description
String's Instance Variables
"Bush"
String's Compiled

`list[0] = c;`


list
c
Array
5
null
null
null
null
Customer's Instance Variables
11
Jeb
Bush
Customer's Description
Customer's Compiled Byte Code
<cinit>()
<init>()
getFirstName()
getLastName()
:
toString()
Customer's Static Variables
11

```
list[1] = new Customer("George","Bush");
```



list

c

Array

5

null

null

null

Customer's Instance Variables

11

Jeb

Bush

Customer's Instance Variables

22

George

Bush

Customer's Description

Customer's Compiled Byte Code

<cinit>()

<init>()

getFirstName()

getLastName()

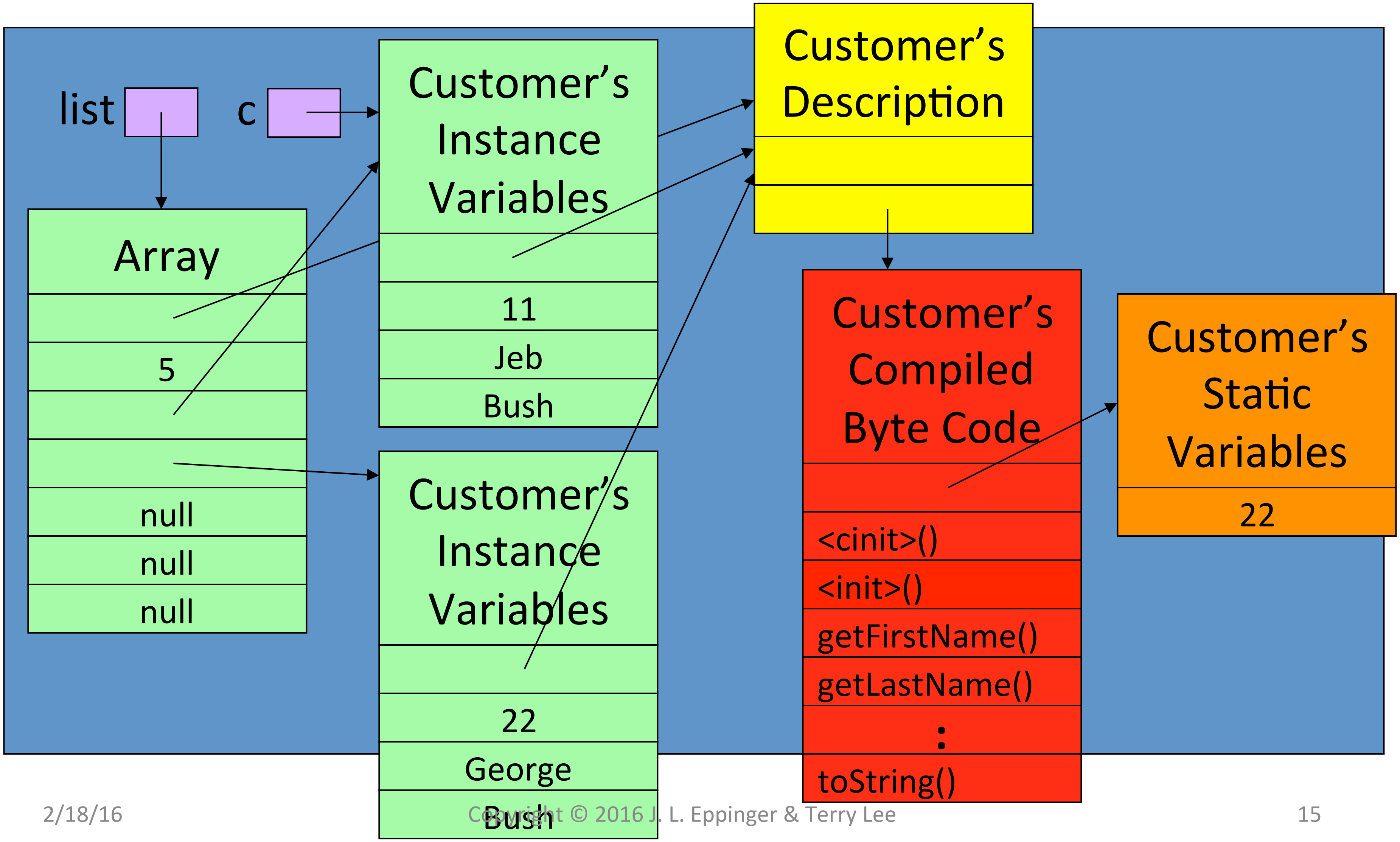
:

toString()

Customer's Static Variables

22

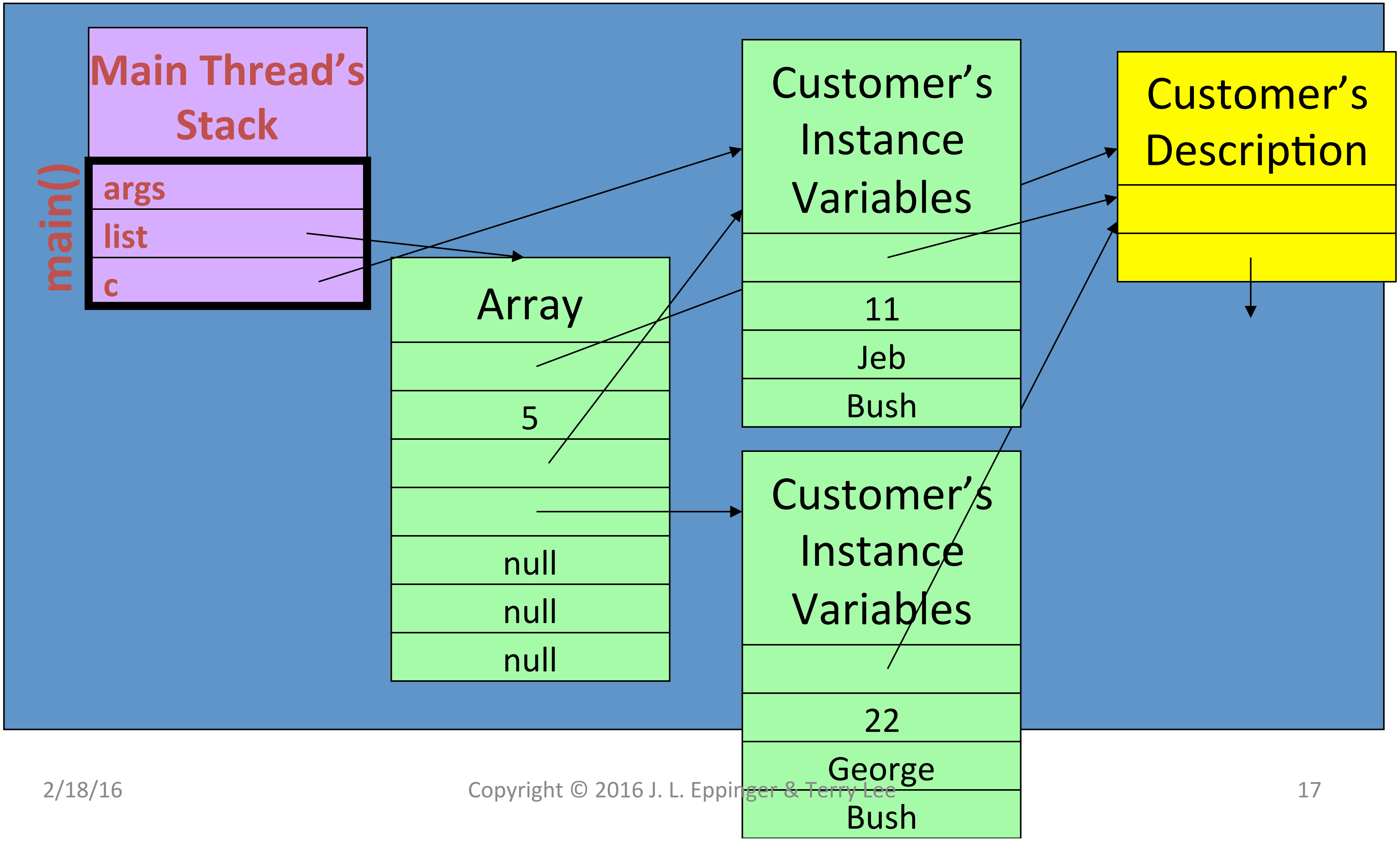
Where are the Local Variables kept?



Remember something like this too? (lecture 6)

```
public class CustomerTest {
    // method arguments: args
    public static void main(String[] args) {
        // local variables: list, c
        Customer[] list = new Customer[5];
        Customer c = new Customer("Jeb", "Bush");
        list[0] = c;
        list[1] = new Customer("George", "Bush");
    }
}
```

Local Variables & Method Arguments (& retval & retaddr) Kept in Thread Stack



What’s Recursion?

“To iterate is human, to recurse is divine.”

– unknown

“To understand recursion, one must understand recursion”

– anonymous

Factorial

The product of all positive integers less than or equal to *n*

```
public static int factorial(int n) {
    int answer = 1;

    for (int i = 1; i <= n; i++) {
        answer = answer * i;
    }

    return answer;
}
```

factorial(4) : 1 * 2 * 3 * 4 = 24

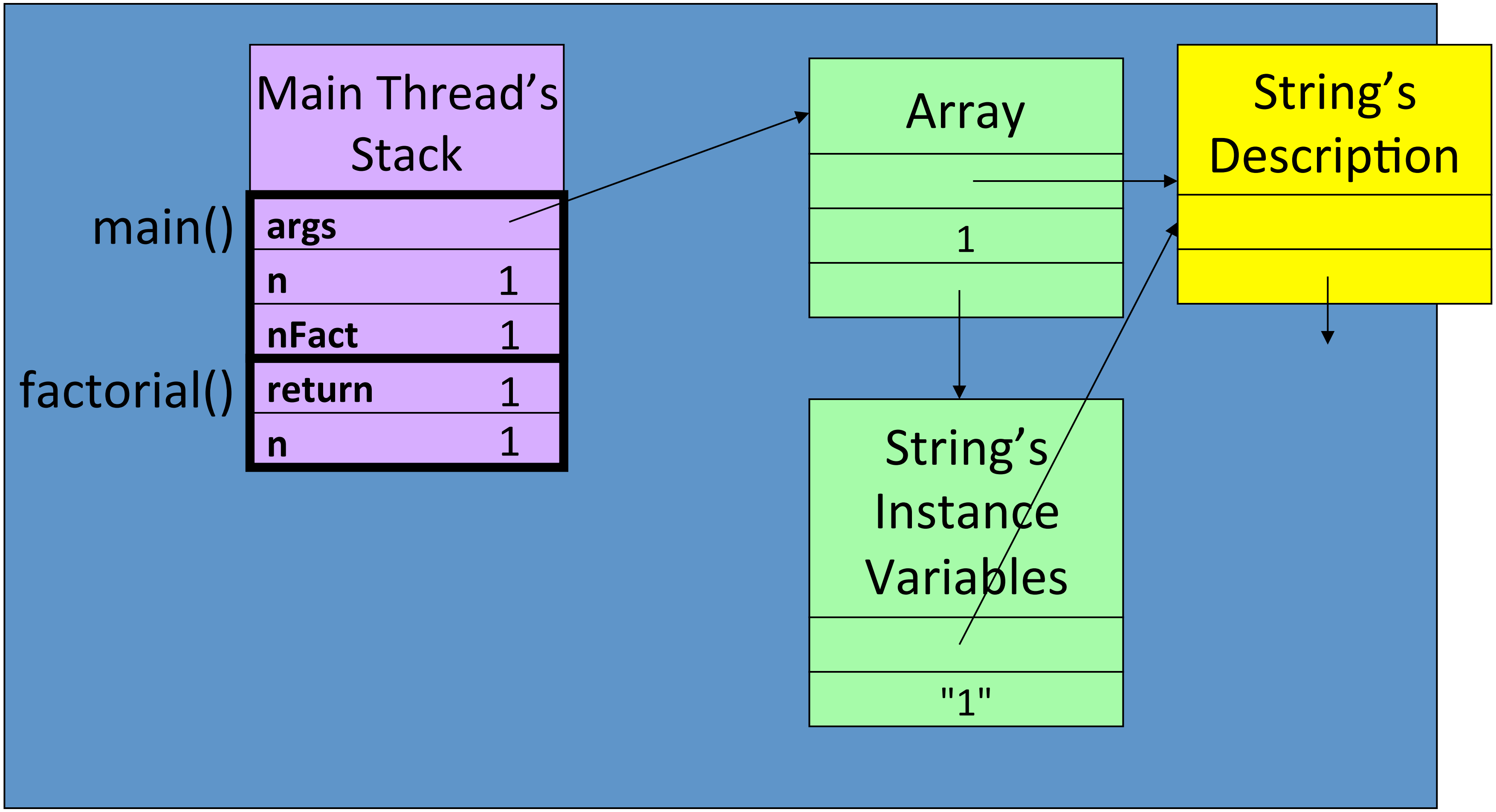
Factorial Recursively

factorial(4) : 1 * 2 * 3 * 4 = 24   -> 4 * factorial(3)
factorial(3) : 1 * 2 * 3 = 6   -> 3 * factorial(2)
factorial(2) : 1 * 2 = 2   -> 2 * factorial(1)
factorial(1) : 1 = 1

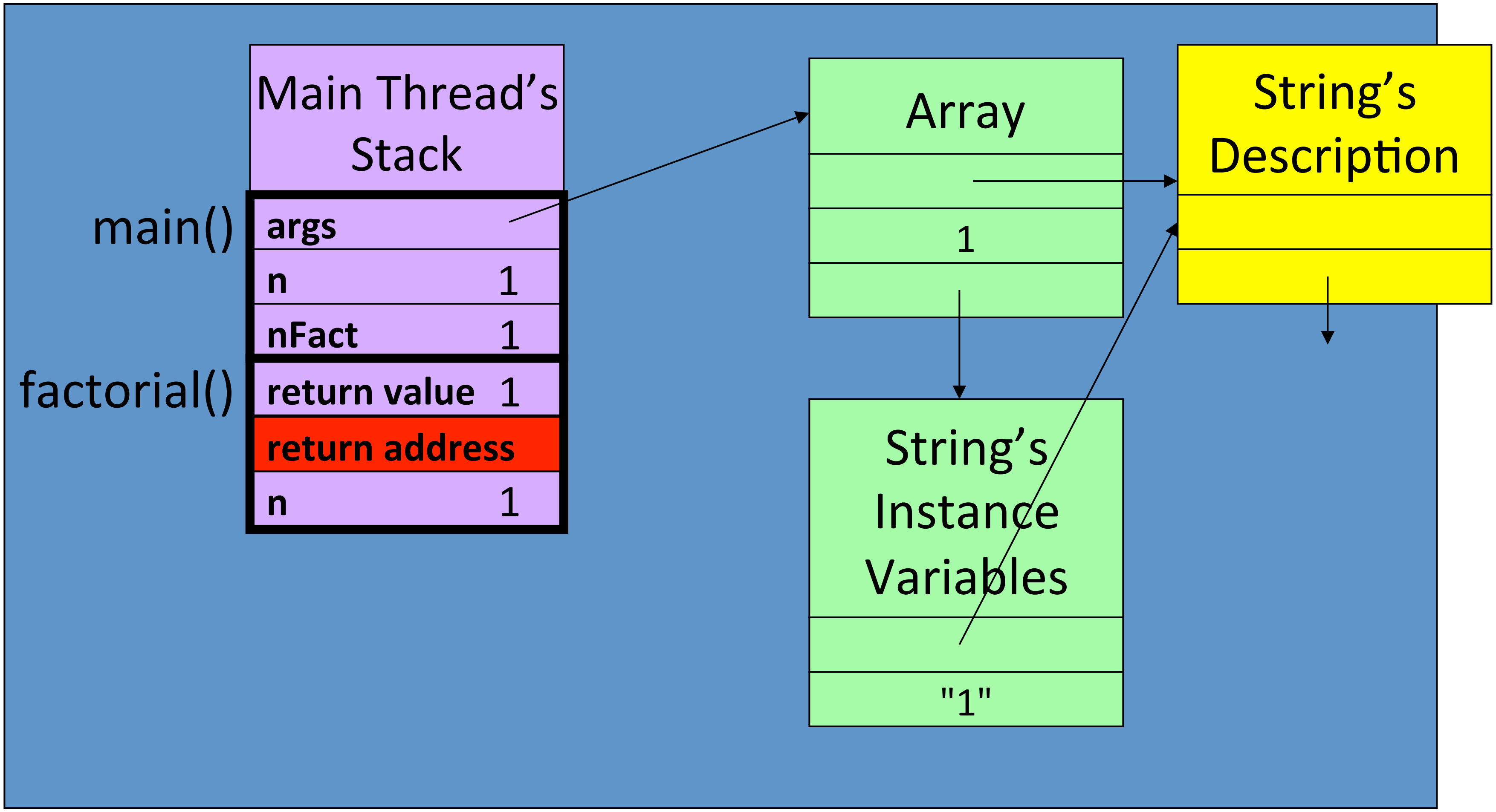
```
public static int factorial(int n) {
    if (n == 1) return 1;
    return n * factorial(n-1);
}
```

- Can you do this in Java?
- How does it work? How does it keep track of return values?

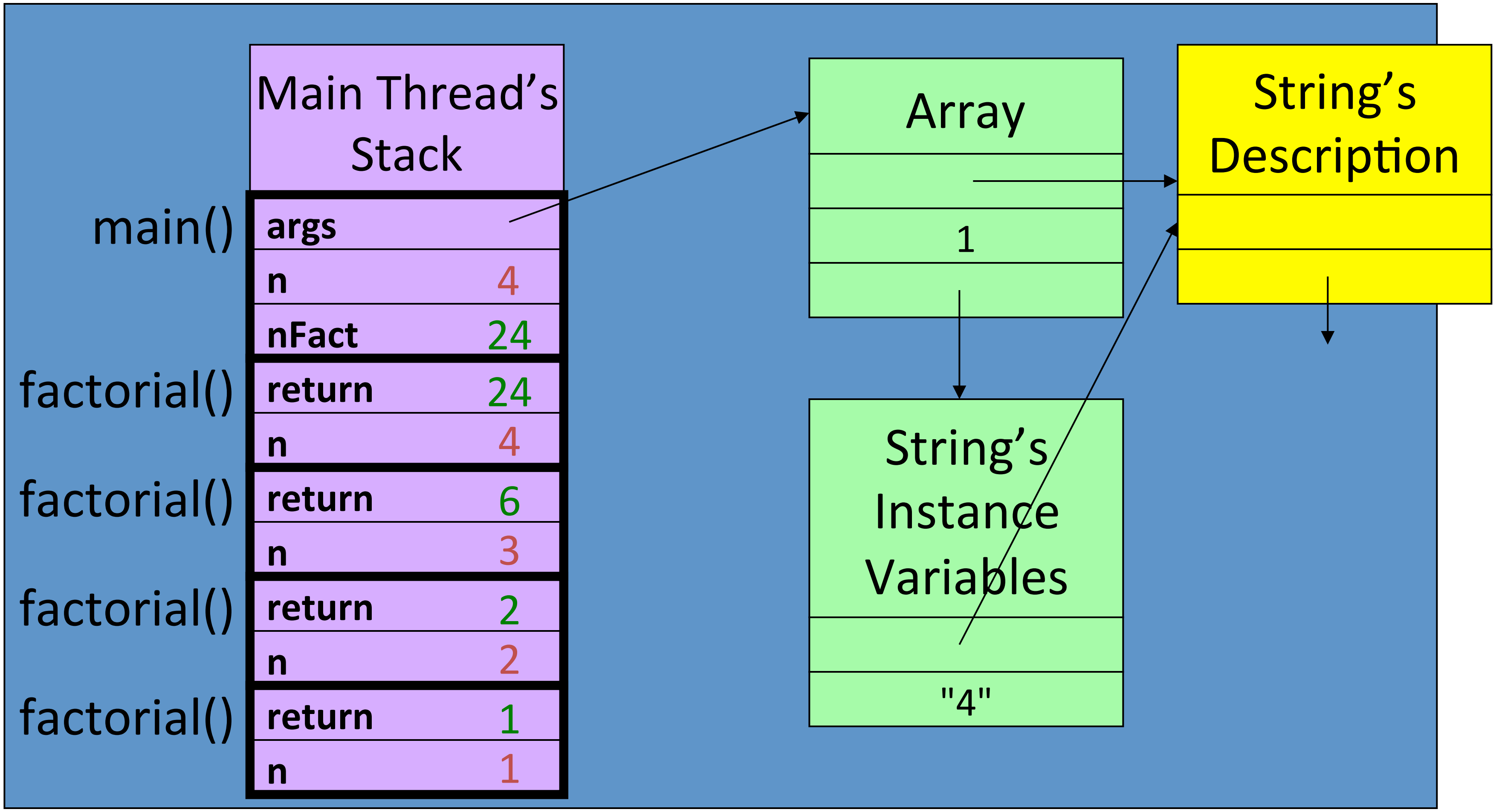
Factorial(1)


Main Thread's Stack
main()
args
n 1
nFact 1
factorial()
return 1
n 1
Array
1
String's Description
String's Instance Variables
"1"

Note: There is a Return Address



factorial(4)


Main Thread's Stack
main()
args
n
4
nFact
24
factorial()
return
24
n
4
factorial()
return
6
n
3
factorial()
return
2
n
2
factorial()
return
1
n
1
Array
1
String's Description
String's Instance Variables
"4"

How to handle exceptional cases?

- What if n <= 0 in factorial example?
  - – Should we pass back 0 or -1?

Throw an Exception

```
public static int factorial(int n) {
    if (n <= 0) {
        throw new IllegalArgumentException(”only posi…
    }

    if (n == 1) return 1;
    return n * factorial(n-1);
}
```

Now, you can try to catch in the main method

Advantages of Exceptions

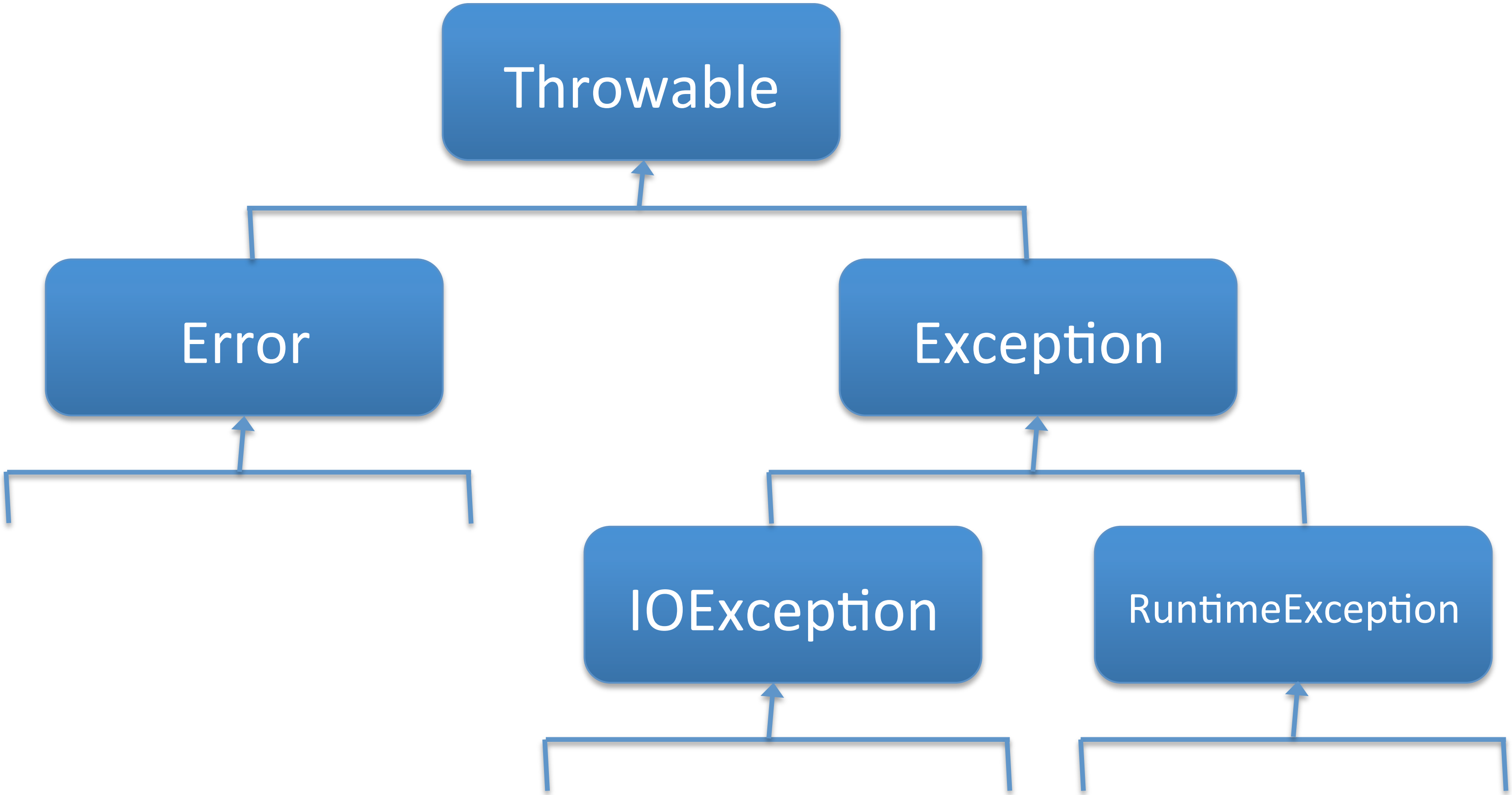
- There may be no obvious way of distinguishing between valid and invalid value.
  - Example: Integer.parseInt()
  - Use/create groups of exceptions and handle exceptions in a general fashion
- Callers do not need to check after every call for error returns
  - It can be done by the caller's caller. If not, eventually JVM catches it
  - Let those methods interested in handling exceptions to handle

  Example: `FactorialTest.java`

Exceptions

- Exceptions are Objects
  - Actually, they are subclasses of `Throwable`
  - Contains:
    - a String message
    - a Throwable cause
    - A Stack Trace

Exception Hierarchy



Checked vs. Unchecked

- Any exception from Error class or RuntimeException is an unchecked exception
  - **`IllegalArgumentException`**
  - **`NumberFormatException`**
  - **`ArrayIndexOutOfBounds`**
  - **`StackOverflowError`**
- All other exceptions are checked exceptions
  - – Because compiler checks whether you handle these or not at compile time
  - – Either try/catch or throw again!
    - **`FileNotFoundException`**
    - **`IOException`**
    - **`InterruptedException`**

Throwing Exceptions

- To throw one, create an instance and say `throw`

```
throw new IllegalArgumentException("Bad…");
throw new NumberFormatException("For input s…");
```

Catching Exceptions

- Use a try/catch statement

```
try {
    factorial(-1);
} catch (Exception e) {
    e.printStackTrace();
}
```

Many Catchers

- Catch clauses are tried in order (**Order matters!**)

```
try {
    factorial(-1);
} catch (NumberFormatException e) {
    System.out.println(…);
    System.exit(1);
} catch (IllegalArgumentException e) {
    System.out.println(…);
    throw e;
}
```

* Specific exceptions should come before general ones

Check out JavaDoc of NumberFormatException

https://docs.oracle.com/javase/8/docs/api/java/lang/NumberFormatException.html

The `finally` clause

- When an exception occurs, it skips the remaining code in the `try` block and exits
- The code in `finally` block executes whether or not an exception was caught
  - Thus, useful for preventing resource leaks such as closing a file, etc.
- You can use the `finally` clause without a `catch` clause

throw vs. throws

- Don’t confuse throw statement with the throws clause!
  - throw statement causes an exception to be thrown in method body

```
throw new IllegalArgumentException(“invalid value”);
```

  - throws clause informs the compiler that a method throws one ore more exceptions in method header

```
public int read() throws IOException {
   …
}
```

If you take 08-672 (J2EE Web App), you will have a lot of fun with exception handlings

Don’t squelch exceptions

```
try {
    dangerousCode();
} catch (Exception e) {
}
```

Don’t squelch exceptions

```
try {
    dangerousCode();
} catch (Exception e) {
    throw new AssertionError(“issue”, e);
}
```

What does this do?

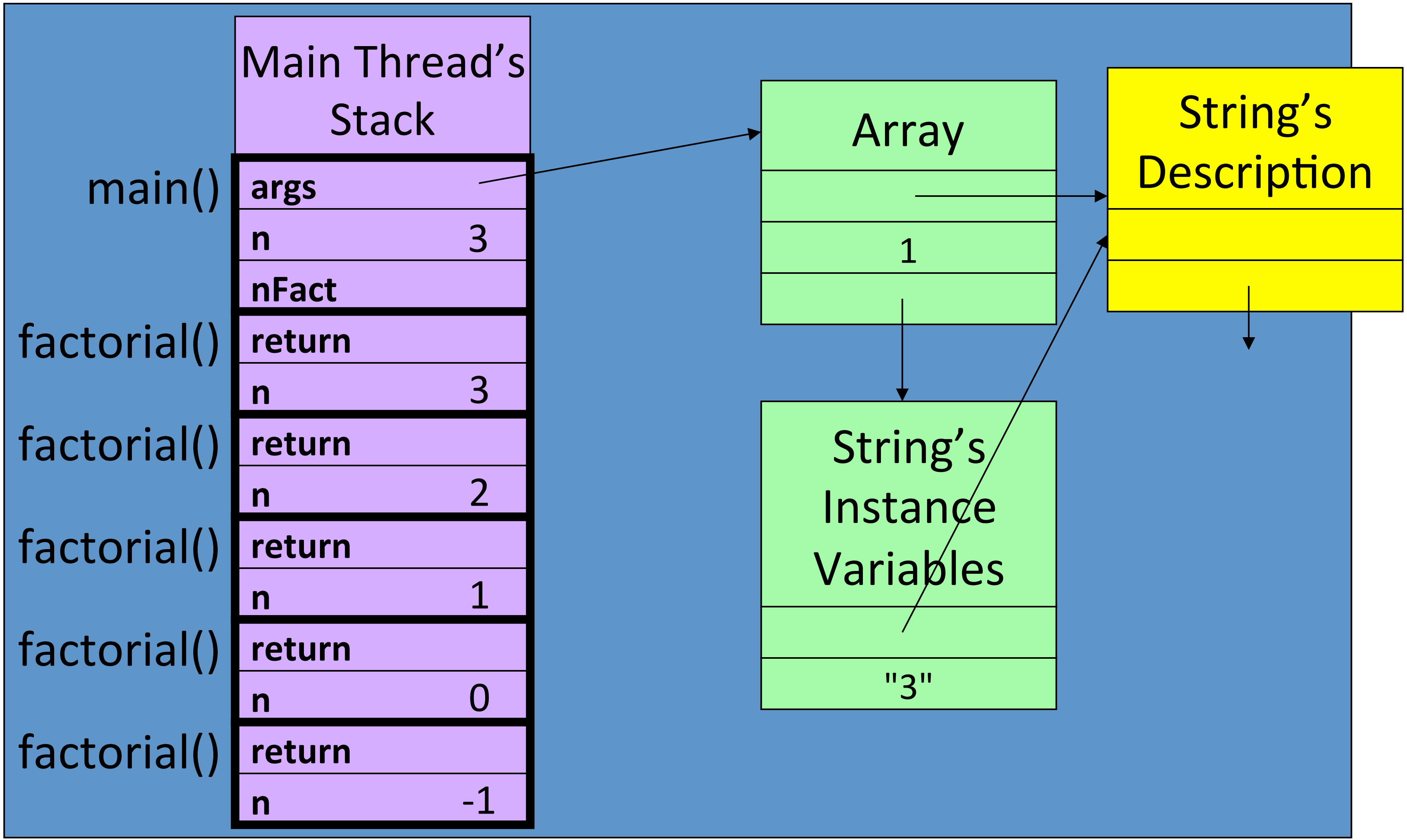
```
public static int factorial(int n) {
    if (n <= 0) {
        throw new IllegalArgumentException(”nly pos…
    }

    return n * factorial(n-1);
}
```

How about this one?

```
public static int factorial(int n) {
    return n * factorial(n-1);
}
```

If you take 08-722 (DSAP), you will have a lot of fun with recursion

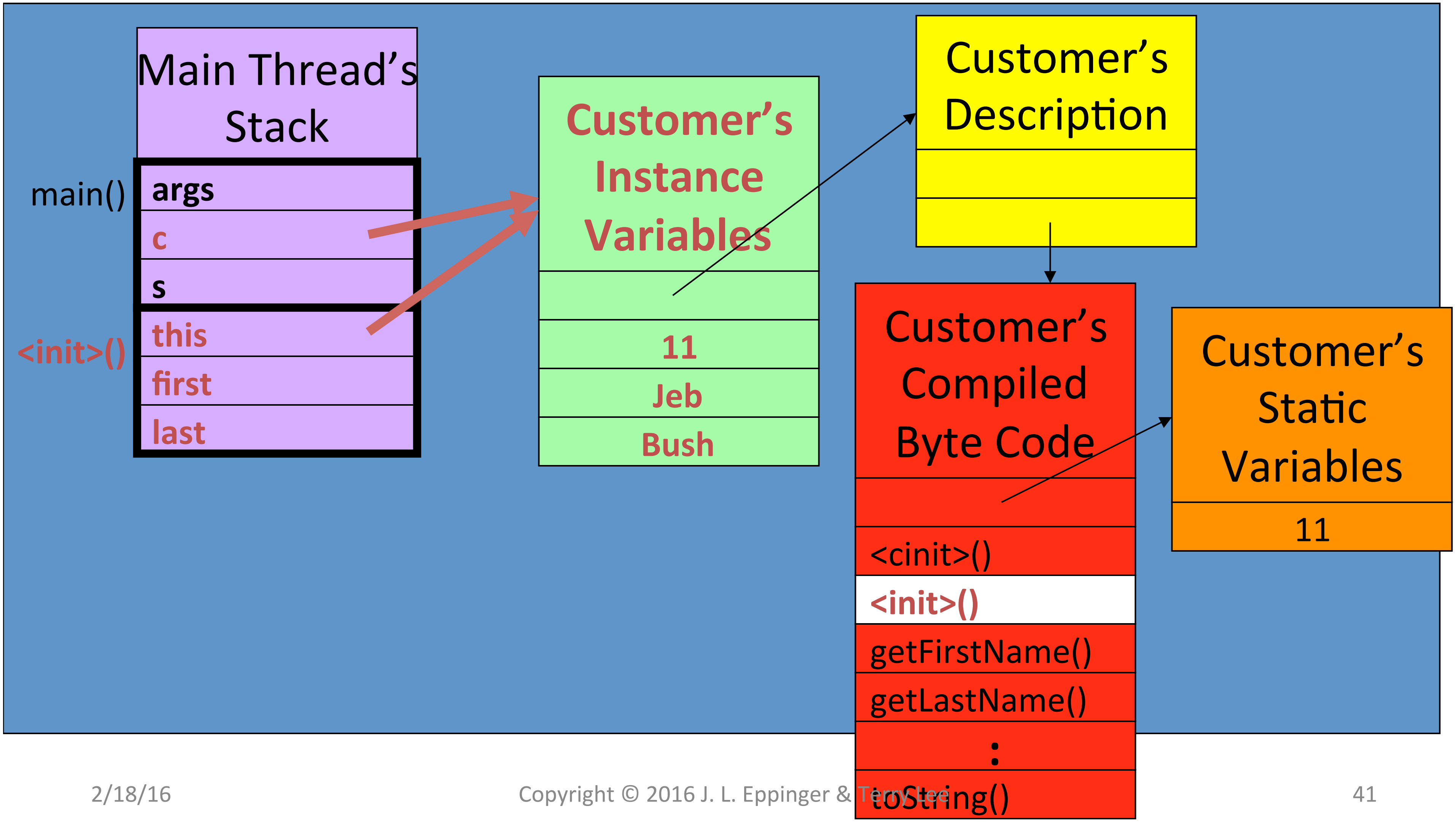
Try it for factorial(3)



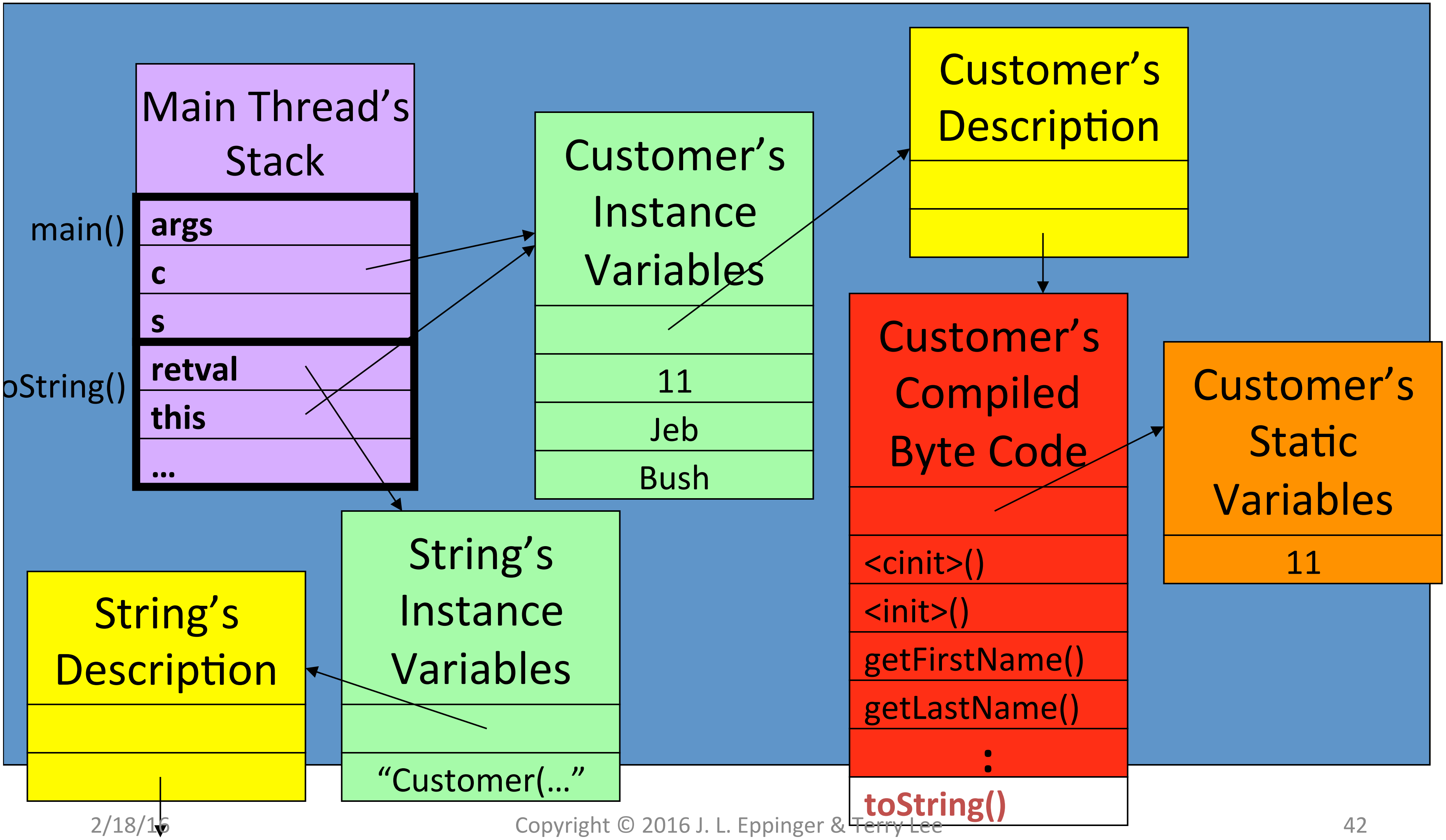
What about Instance Methods?

```
public class CustomerTest {
    public static void main(String[] args) {
        Customer c = new Customer("Jeb","Bush");
        String s = c.toString();
    }
}
```

`Customer c = new Customer("Jeb","Bush");`



String s = c.toString();



Main Thread's Stack

main()

args

c

s

oString()

retval

this

…

Customer's Instance Variables

11

Jeb

Bush

Customer's Description

Customer's Compiled Byte Code

<cinit>()

<init>()

getFirstName()

getLastName()

:

toString()

Customer's Static Variables

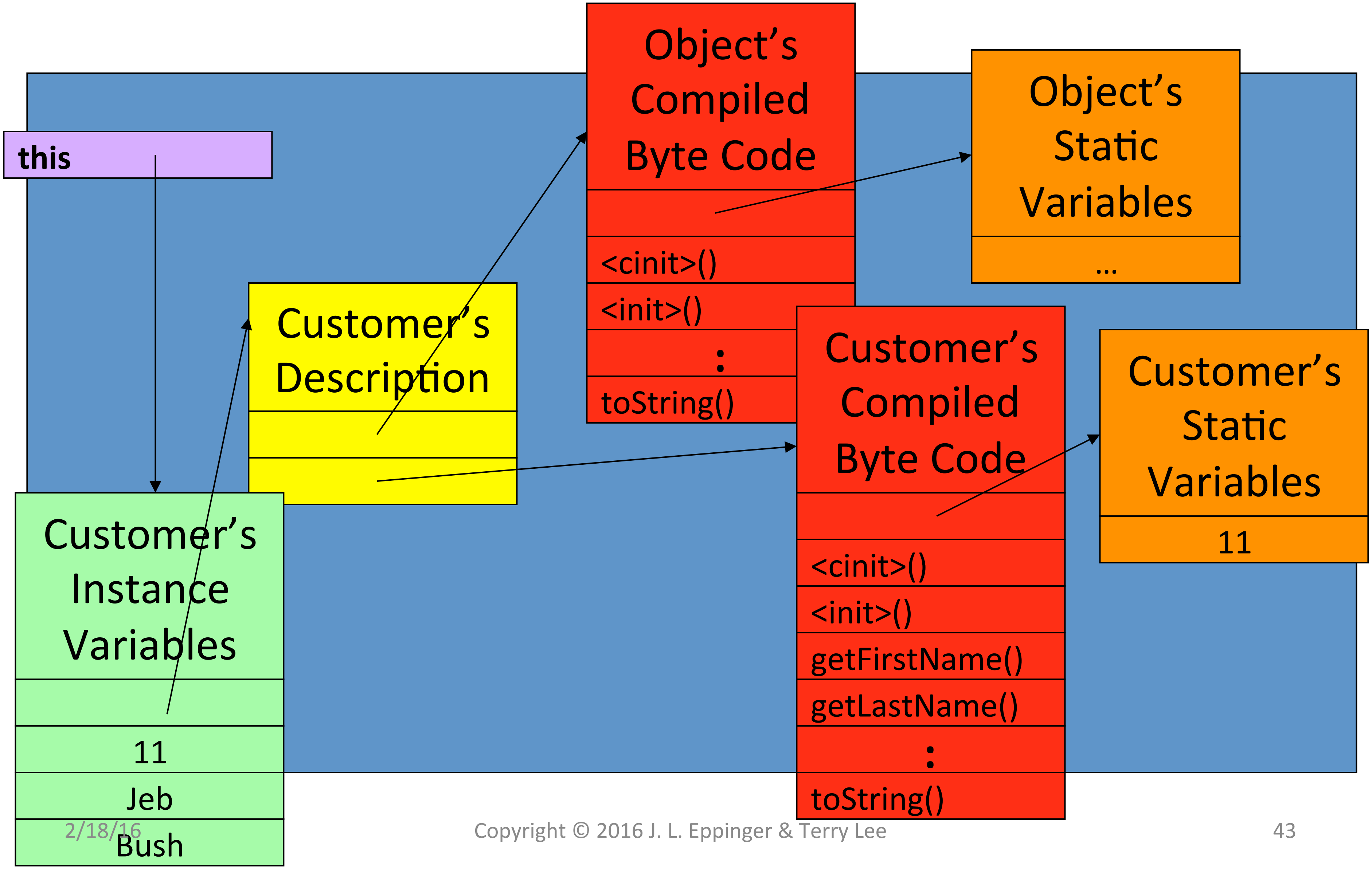
11

String's Instance Variables

"Customer(…"

String's Description

`c.toString()` and Class Hierarchies


this
Object's Compiled Byte Code
<cinit>()
<init>()
:
toString()
Object's Static Variables
…
Customer's Description
Customer's Compiled Byte Code
<cinit>()
<init>()
getFirstName()
getLastName()
:
toString()
Customer's Static Variables
11
Customer's Instance Variables
11
Jeb
Bush

Remember this? (about `this` keyword)

- A **reference to the current object**
- Because a field is shadowed by a method or constructor parameter

```
public class Point {
    private int x = 0;
    public Point(int x) { this.x = x; }
}
```

- To call another constructor in the same class

```
public Rectangle() {
    this(1, 1);
}
public Rectangle(int width, int height) {
    …
}
```

Use `this` to reference instance variables

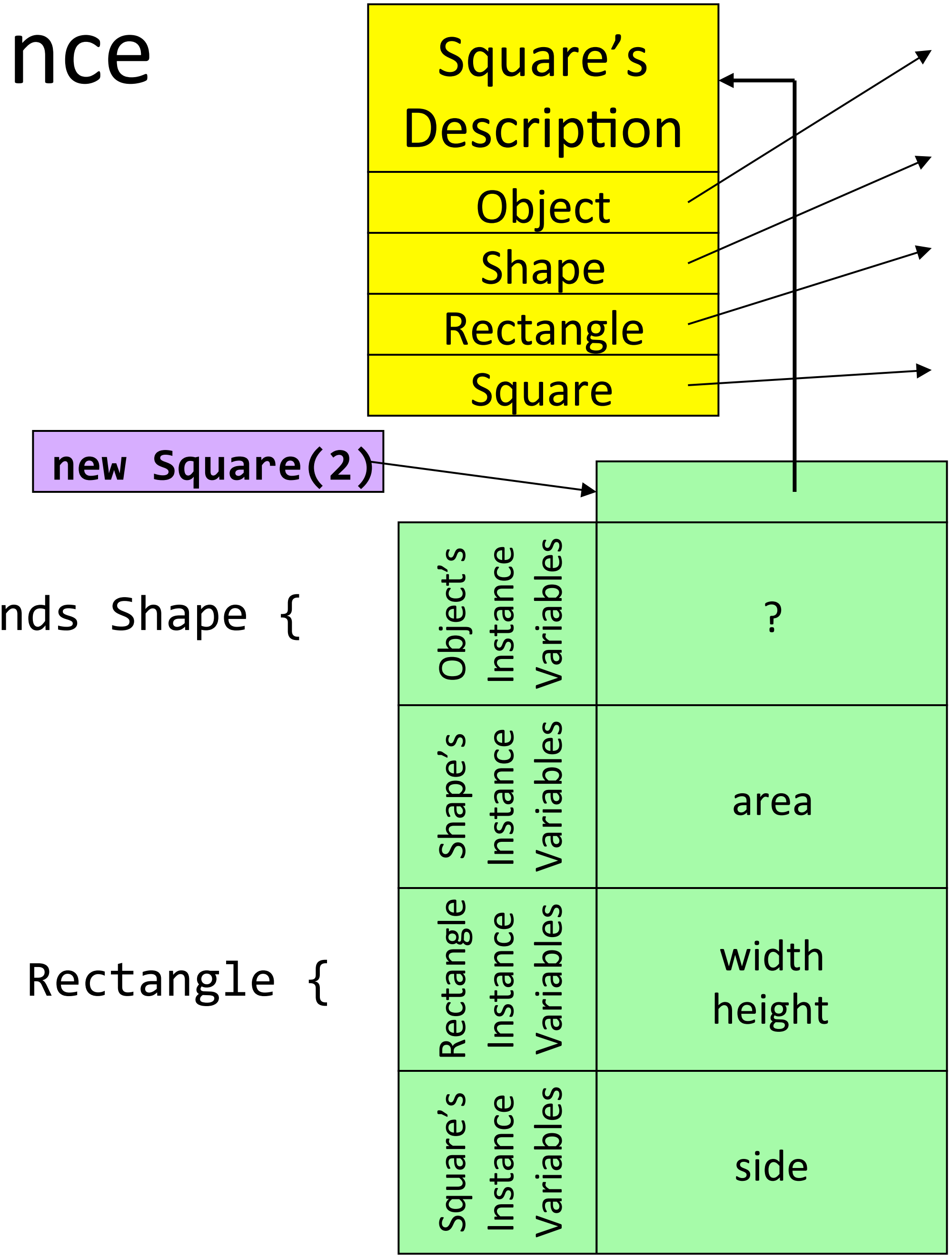
```
private String firstName;
private String lastName;

public Customer(String first, String last) {
    this.firstName = first;
    this.lastName  = last;
}
```

Use **`this`** to refer to current instance

```
JButton b = new JButton("Click Me");
b.addActionListener(this);
```

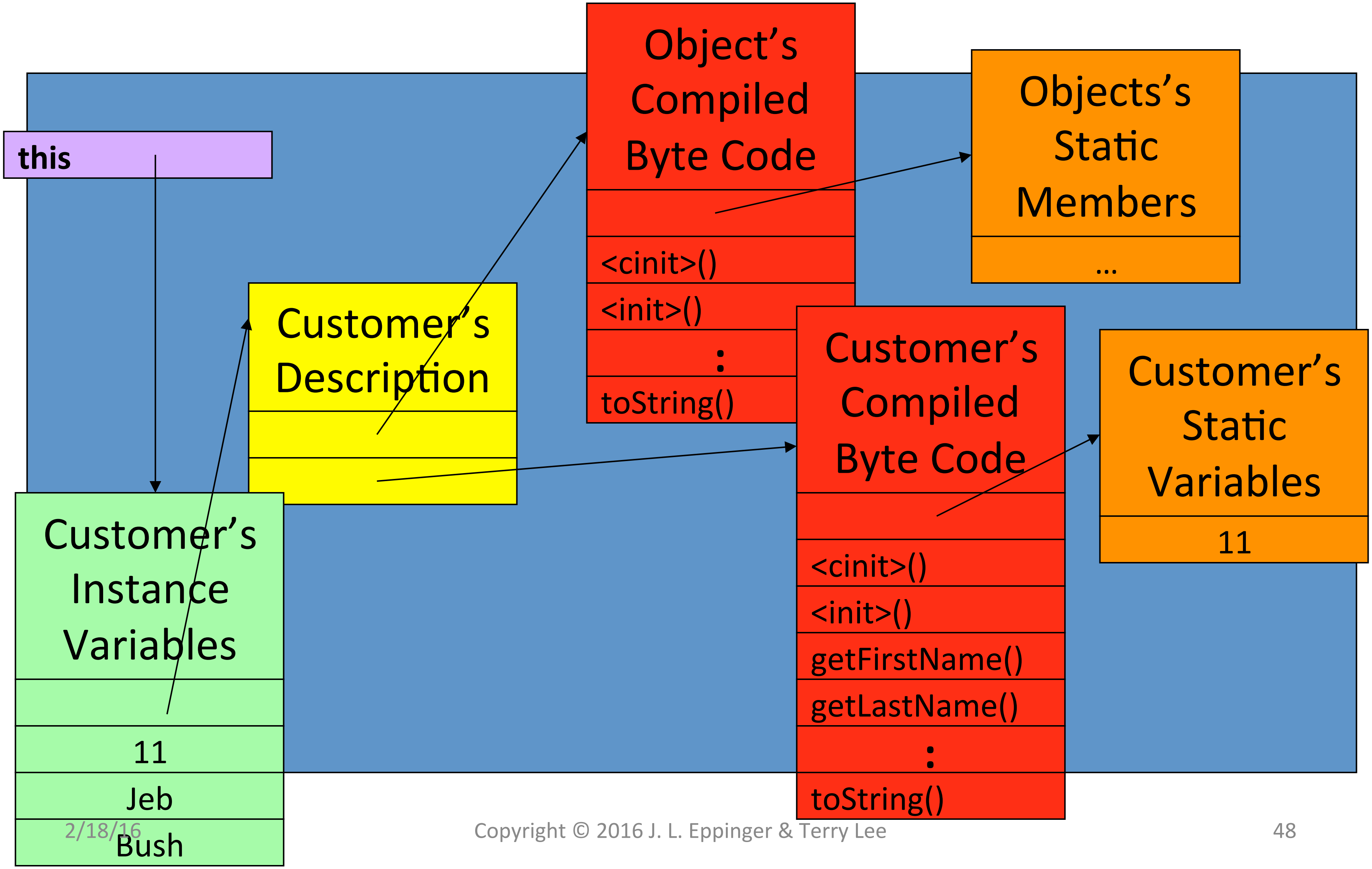
Superclass Instance Variables

```
public class Shape {
    protected double area;
}

public class Rectangle extends Shape {
    private double width;
    private double height;
}

public class Square extends Rectangle {
    private double side;
}
```



Object's Instance Variables	?
Shape's Instance Variables	area
Rectangle Instance Variables	width height
Square's Instance Variables	side

How about super?



How about super?

```
public class MyClass extends OtherClass {
    private String myField;
    public MyClass(String a, String b, String c) {
        super(a,b);
        myField = c;
    }
    public String toString() {
        return super.toString() + "," + myField;
    }
}
```

- **super keyword is not really the same as this keyword**

  this is a reference to the current instance but super is not a reference!

Casting

- When an “Object” is returned to you, you may need to cast it

```
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    JButton button = (JButton) source;
    …
}
```

casting

Implicit vs. Explicit Casting

- Casting is not required when going up the class hierarchy
  - Sometimes called "Implicit" casting
- Casting is required when going down the class hierarchy
  - This is called "explicit" casting
  - Java can (sometimes) tell an explicit cast is not possible
    - Example: You cannot cast a String into a Date

Casting vs. Conversion

- When converting between numeric primitive types, we use the same "cast" construct
  - And we say "cast"
- But, this is technically type "conversion"
  - You're not treating the same reference as a different type
  - You are converting the value of the primitive type into a different primitive type
- Example:

```
JButton button = (JButton) source;
```

vs

```
double d = 2.5 * 3.5;
int x = (int) d;
```

Outline

✓Course Plan

✓Questions

✓Memory Upgrade

✓Recursion

✓Exceptions

✓More on Classes (`this`, `super` & casting)

⇢More on Threads

Questions

Threads & Memory

- **Each thread has its own stack!**
- Each thread has its own local variables
  - Primitive types are clearly not shared
  - References to objects are not shared
    - Objects may be shared!!
- Each object instance has its own instance variables

Outline

✓ Course Plan

✓ Questions

✓ Detailed Picture of Memory

✓ Recursion

✓ Exceptions

✓ More on Classes (`this`, `super` & casting)

✓ More on Threads

⇢ Questions

How big are factorials?

- What's the biggest factorial that fits in an int?
- How can you do better?

How to find more about Class at runtime?

- `java.lang.Class`
  - provides methods to examine the runtime properties of the object including its members and type information
  - Along with classes and interfaces in java.lang.reflect package, use it to analyze capabilities of classes at runtime (Reflection)

* Example : `CustomerInspector.java`

Next Week

- Java 8
- More about Interfaces
- Functional Programming
- Written Exam