Introduction to Java

Dan S. Wallach and Mack Joyner, Rice University
Reminder: Fill out our web form!

Fill this out ASAP if you haven’t already.
http://goo.gl/forms/ArYKWBc0zY

We need this to connect you to Subversion and Piazza.
You need that for project 1. Due Sunday night!
Java Types

Java has 8 primitive data types

- boolean
- char
- Integers: byte, short, int, long
- Floating point numbers: float, double

Java also has non-primitive data types
Classes, Arrays

Default value is 0
Default value is null
public class LISubsequence {
    private int[] a;

    public LISubsequence() {
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        LISubsequence lis = new LISubsequence();
        lis.a~ = ...  
    }
}

Variables: Instance (Global) vs Local

public class LISubsequence {
    private int[] a;

    public LISubsequence() {
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        LISubsequence lis = new LISubsequence();
        lis.a~ = ...  
    }
}
Variables: Instance (Global) vs Local

```java
public class LISubsequence {
    private int[] a;

    public LISubsequence() {
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        LISubsequence lis = new LISubsequence();
        lis.a = ...  // Not legal, a is private
    }
}
```

In Python, global variables can be here

```python
global variable (field) a
```

In Java, variables must be declared within a scope (class or method) and can be accessed within that scope.

```
private int[] a;
```
Constructors and methods may have parameters.

Use “this.” to reference member field.

```
public class LISubsequence {
    private int[] a;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}
```
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}

int[] a: 1-D int array

Each element must have type int.

array type: []
In Java, method calls are call-by-value.

Pass value of variable to call
**Method Calls: Call-By-Value**

In Java, method calls are call-by-value.

Pass value of variable to call

```java
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a2) {
        this.a = a2;
    }
    public void findLIS(int[] b) {
        b = new int[]{2, 3, 5, 6};
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a1);
        lis.findLIS(a1);
    }
}
```
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a2) {
        this.a = a2;
    }
    public void findLIS(int[] b) {
        b = new int[]{2, 3, 5, 6};
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a1);
        lis.findLIS(a1);
    }
}

In Java, method calls are call-by-value.
Pass value of variable to call
In Java, method calls are call-by-value.

Pass value of variable to call
**Method Calls: Call-By-Value**

In Java, method calls are call-by-value. Pass value of variable to call.

```java
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a2) {
        this.a = a2;
    }
    public void findLIS(int[] b) {
        b = new int[]{2, 3, 5, 6};
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a1);
        lis.findLIS(a1);
    }
}
```
In Java, method calls are call-by-value.

Pass value of variable to call

```java
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a2) {
        this.a = a2;
    }
    public void findLIS(int[] b) {
        b = new int[]{2, 3, 5, 6};
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a1);
        lis.findLIS(a1);
    }
}
```

a1[0] is still 1 after call to findLIS
In Java, method calls are call-by-value.

Pass value of variable to call

```java
public class LISubsequence {
    private int[] a;
    public LISubsequence(int[] a2) {
        this.a = a2;
    }
    public void findLIS(int[] b) {
        b = new int[]{2, 3, 5, 6};
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a1);
        lis.findLIS(a1);
    }
}```
Get started! Due Sunday night.
public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}

The **final** keyword:

primitive type - value can’t change

non-primitive type - ref can’t change but contents of ref object can.
Which assignments in findLIS are legal?

public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
        a = new int[]{2, 3, 5};
        a[0] = 2;
        lbound = -2;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}

The **final** keyword:

primitive type - value can’t change

non-primitive type - ref can’t change but **contents** of ref object can.
Which assignments in `findLIS` are legal?

```java
public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
        a = new int[]{2, 3, 5};
        a[0] = 2;
        lbound = -2;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}
```
Which assignments in findLIS are legal?

```java
public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
        a = new int[]{2, 3, 5};
        a[0] = 2;
        lbound = -2;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}
```

Not legal, ref changed
Which assignments in findLIS are legal?

```java
public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
        a = new int[]{2, 3, 5};
        a[0] = 2;
        lbound = -2;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}
```

Legal, ref didn’t change
Which assignments in findLIS are legal?

```java
public class LISubsequence {
    private final int[] a;
    private final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
        a = new int[] {2, 3, 5};
        a[0] = 2;
        lbound = -2;
    }
}
```

```java
public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a = {1, 3, 5};
        LISubsequence lis = new LISubsequence(a);
    }
}
```

Not legal, value changed.
public class LISubsequence {
    private final int[] a;
    public static final int lbound = -1;

    public LISubsequence(int[] a) {
        this.a = a;
    }

    public void findLIS() {
        int index = 0;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {1, 3, 5};
        LISubsequence lis1 = new LISubsequence(a1);
        LISubsequence lis2 = new LISubsequence(a2);
    }
}
public class LISubsequence {

    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {4, 6, 8};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.findLIS(a2);
    }
}

Only `findLIS` method refs `a`.

Eliminate member field `a` and `lbound`.

Eliminated need for second object instance.
public class LISubsequence {
    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;

        // iterate through every element in array
        for (int num : a) {
            index++;
        // increment index by 1
        }
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {4, 6, 8};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.findLIS(a2);
    }
}
public class LISubsequence {

    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int highNum = -1;

        //iterate through every element in array
        for (int i = 0; i < a.length; i++) {
            int num = a[i];
            int num = a[i];
        }
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {4, 6, 8};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.findLIS(a2);
    }
}
public class LISubsequence {

    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int highNum = -1;

        // iterate through odd array indices
        for (int i = 1; i < a.length; i+=2) {
            int num = a[i];
            highNum = Math.max(highNum, num);
        }
    }

    public class LISubsequenceTest {

        public void testFindLIS() {
            int[] a1 = {1, 3, 5};
            int[] a2 = {4, 6, 8};
            LISubsequence lis = new LISubsequence();
            lis.findLIS(a1);
            lis.findLIS(a2);
        }
    }
}

Iterate through array subset
Array index out of bounds exception possible
public class LISubsequence {

    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;

        //iterate through every element in array
        while (index < a.length) {
            int num = a[index];
            index++;
        }
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {4, 6, 8};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.findLIS(a2);
    }
}

Execute while loop body each time while loop condition is true.
public class LISubsequence {

    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;

        //iterate through every element in array
        for (int num : a) {
            if (num > highNum) {
                highNum = num;
            }
            index++; //increment index by 1
        }
    }
}

public class LISubsequenceTest {

    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        int[] a2 = {4, 6, 8};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.findLIS(a2);
    }
}
import java.util.LinkedList;

public class LISubsequence {
    public LISubsequence() {
    }
    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;
        LinkedList l = new LinkedList();

        //iterate through every element in array
        for (int num : a) {
            if (num > highNum) {
                l.addLast(index);
                highNum = num;
            }
            index++; //increment index by 1
        }
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
    }
}

LinkedList includes:

- **addLast**(Object o) - add o to end of list
- **addAll**(Collection c) - add all objects in c to end of list
- **size**() - number of elements in list
- **toString**() - represent list as String
Mutation!
import java.util.LinkedList;
public class LISubsequence {

    public LISubsequence() { }

    public void findLIS(int[] a2) {
        int index = 0;
        int highNum = -1;
        LinkedList l = new LinkedList();

        //iterate through every element in array
        for (int num : a2) {
            if (num > highNum) {
                a2[index] = index;
                highNum = num;
            }
            index++; //increment index by 1
        }
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {4, 5, 2};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.fastFindLIS(a1);
    }
}
import java.util.LinkedList;
public class LISubsequence {
    public LISubsequence() {
    }

    public void findLIS(int[] a2) {
        int index = 0;
        int highNum = -1;
        LinkedList l = new LinkedList();

        // iterate through every element in array
        for (int num: a2) {
            if (num > highNum) {
                a2[index] = index;
                highNum = num;
            }
            index++;  // increment index by 1
        }
    }...
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {4, 5, 2};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.fastFindLIS(a1);
    }
}
import java.util.LinkedList;
public class LISubsequence {

    public LISubsequence() {} 

    public void findLIS(int[] a2) {
        int index = 0;
        int highNum = -1;
        LinkedList l = new LinkedList();

        //iterate through every element in array
        for (int num:a2) {
            if (num > highNum) {
                a2[index] = index;
                highNum = num;
            }
            index++; //increment index by 1
        }
    }

    public class LISubsequenceTest {
        public void testFindLIS() {
            int[] a1 = {4, 5, 2};
            LISubsequence lis = new LISubsequence();
            lis.findLIS(a1);
            lis.fastFindLIS(a1);
        }
    }
}

Be careful. Mutation may cause unwanted side-effects.

Oops, made a mistake

Memory

a1

{0,1,2}

a2
import java.util.LinkedList;

public class LISubsequence {
    public LISubsequence() {
    }

    public void findLIS(int[] a2) {
        int index = 0;
        int highNum = -1;
        LinkedList l = new LinkedList();

        // iterate through every element in array
        for (int num : a2) {
            if (num > highNum) {
                a2[index] = index;
                highNum = num;
            }
            index++; // increment index by 1
        }
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {4, 5, 2};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
        lis.fastFindLIS(a1);
    }
}

Be careful. Mutation may cause unwanted side-effects.

Oops, made a mistake
Oops, a1 = {0,1,2}
Generics

```java
import java.util.LinkedList;
public class LISubsequence {
    public LISubsequence() {
    }

    public void findLIS(int[] a) {
        int index = 0;
        int highNum = -1;
        LinkedList<Integer> l = new LinkedList<>();
        // iterate through every element in array
        for (int num : a) {
            if (num > highNum) {
                l.addLast(index);
                highNum = num;
            }
            index++;
        }
    }
}

public class LISubsequenceTest {
    public void testFindLIS() {
        int[] a1 = {1, 3, 5};
        LISubsequence lis = new LISubsequence();
        lis.findLIS(a1);
    }
}
```

By default, LinkedList elements have type Object.

A generic type can restrict LinkedList element type.

Compile-time error now occurs when adding non-Integer objects.
Live coding demo

Get easy unit tests to pass for LIS project 1
Get started! Due Sunday night.