Introduction to Java (cs2514)

Lecture 5: Designing Classes

M. R. C. van Dongen

January 30, 2017

Objectives

- Study methods and why they are useful.
- Re-visit the *pass-by-value rule*.
  - Describes how methods should be evaluated.
- Simulate method evaluation.
- Learn how to write classes by carrying out a case study.

Why do we Need Methods?

- Methods are interfaces of parameterised computations.
- Method calls provide reusable computations.
- Building blocks of complex computations.
- Calls are the only mechanism to change `private` variables.

Parameter Taxonomy

Formal parameter: A parameter in a method definition.

Java

```
⟨visibility modifier⟩ ⟨static option⟩
⟨type⟩ ⟨method name⟩( ⟨type_1⟩ ⟨formal parameter_1⟩,
                     ...,
                     ⟨type_n⟩ ⟨formal parameter_n⟩ ) {
    ⟨body⟩
}
```

Actual parameter: A parameter in a method call.

Java

```
⟨reference⟩.⟨method name⟩( ⟨actual parameter_1⟩,
                           ...,
                           ⟨actual parameter_n⟩ );
```

Parameter Taxonomy

Formal parameter: A parameter in a method definition.

Java

```
⟨visibility modifier⟩ ⟨static option⟩
⟨type⟩ ⟨method name⟩( ⟨type_1⟩ ⟨formal parameter_1⟩,
                     ...,
                     ⟨type_n⟩ ⟨formal parameter_n⟩ ) {
    ⟨body⟩
}
```

Actual parameter: A parameter in a method call.

Java

```
⟨reference⟩.⟨method name⟩( ⟨actual parameter_1⟩,
                           ...,
                           ⟨actual parameter_n⟩ );
```

Parameter Taxonomy

Formal parameter: A parameter in a method definition.

Java

```
⟨visibility modifier⟩ ⟨static option⟩
⟨type⟩ ⟨method name⟩( ⟨type_1⟩ ⟨formal parameter_1⟩,
                     ...,
                     ⟨type_n⟩ ⟨formal parameter_n⟩ ) {
    ⟨body⟩
}
```

Actual parameter: A parameter in a method call.

Java

```
⟨reference⟩.⟨method name⟩( ⟨actual parameter_1⟩,
                           ...,
                           ⟨actual parameter_n⟩ );
```

Example

Formal parameters:

Java

```
public void makeMove( final int row, final int column, final String symbol ) {
   :
   :
}
```

Actual parameters:

Java

```
makeMove( row, column, symbol );
:
:
player.makeMove( 0, 0, "x" );
:
:
```

Example

Formal parameters:

Java

```
public void makeMove( final int row, final int column, final String symbol ) {
   :
   :
}
```

Actual parameters:

Java

```
makeMove( row, column, symbol );
:
:
player.makeMove( 0, 0, "x" );
:
:
```

Example

Formal parameters:

Java

```
public void makeMove( final int row, final int column, final String symbol ) {
   :
   :
}
```

Actual parameters:

Java

```
makeMove( row, column, symbol );
:
:
player.makeMove( 0, 0, "x" );
:
:
```

Actual Parameters

Java

```
System.out.println( "The answer is " + 42 + "." );
```

Actual Parameters are Expressions

Java

```
System.out.println( "The answer is " + 42 + "." );
```

Example

Java

```
public static int f( int a, int b ) {
    return a + b;
}

public static void g( int c ) {
    int a = f( 1, 2 + c );
    int d = f( 1 + 3, a );
}
```

Example: Formal Parameters

Java

```
public static int f( int a, int b ) {
    return a + b;
}

public static void g( int c ) {
    int a = f( 1, 2 + c );
    int d = f( 1 + 3, a );
}
```

Example: Actual Parameters

Java

```
public static int f( int a, int b ) {
    return a + b;
}

public static void g( int c ) {
    int a = f( 1, 2 + c );
    int d = f( 1 + 3, a );
}
```

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

The Pass-by-Value Mechanism

Carrying out a Call with n Parameters

1. Create a fresh variable for each parameter.
2. For i from 1 to n (from left to right):
   1. Evaluate the ith actual parameter.
   2. Assign the result of the ith evaluation to the ith fresh variable.
3. Carry out statements in the method body.
4. Return result (if any).
5. Remove fresh variables.

Storing the Value of a Temporary Variable

The Stack

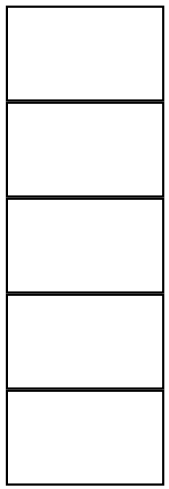
- Actual parameter values are stored on the *stack*.
  - When method is called, variables are created on top of stack.
  - When method returns this scratch space is released.
- The stack also stores the values of local variables in blocks.
  - When block is entered, variable are created on top of the stack.
  - When control leaves the block this scratch space is released.

Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```



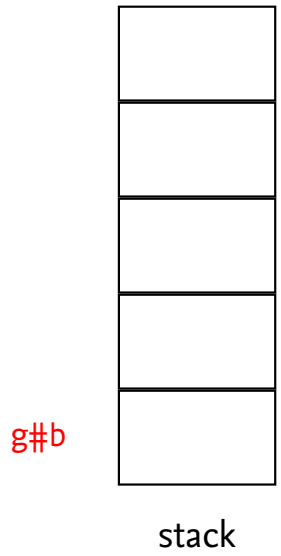
stack

Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

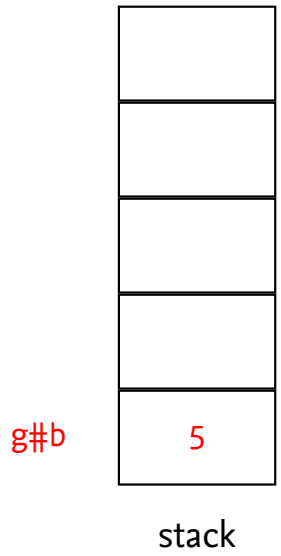


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

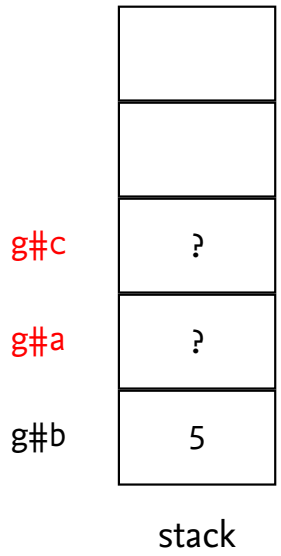


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

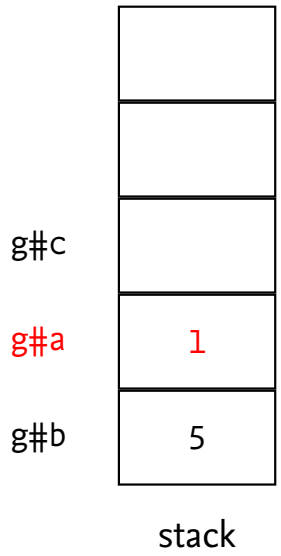


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

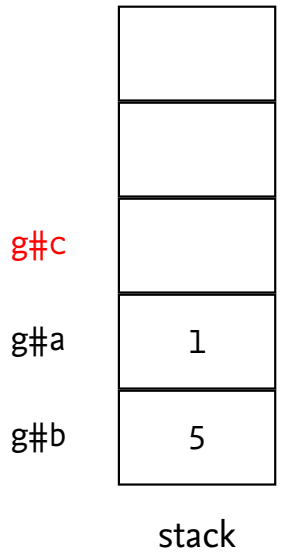


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

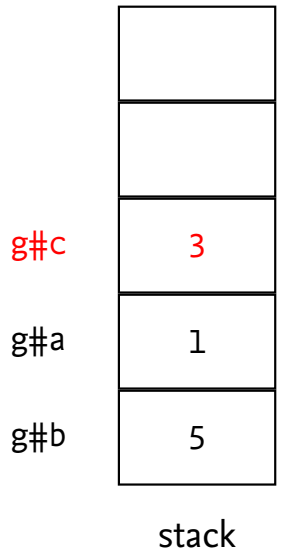


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

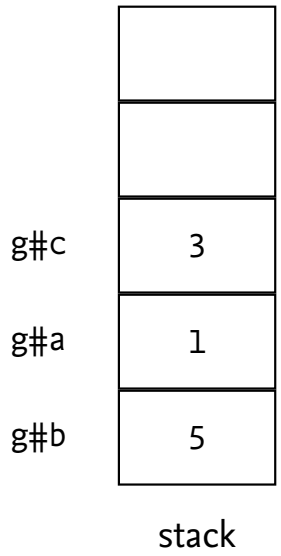


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

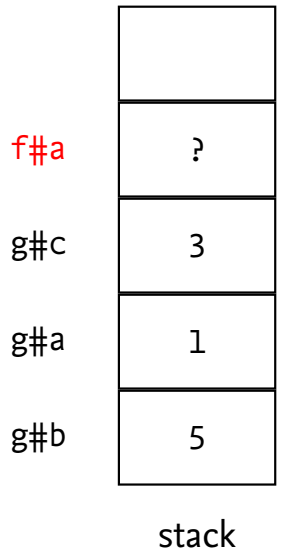


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

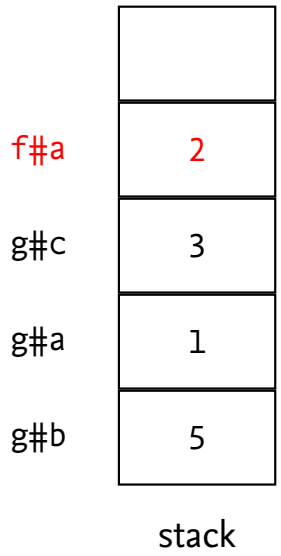


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

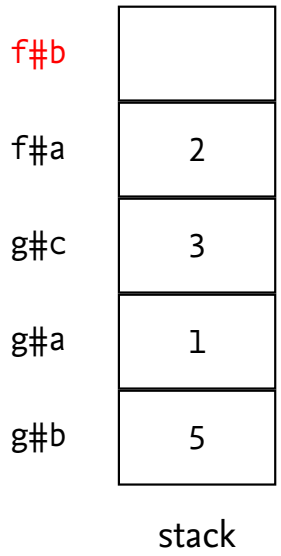


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

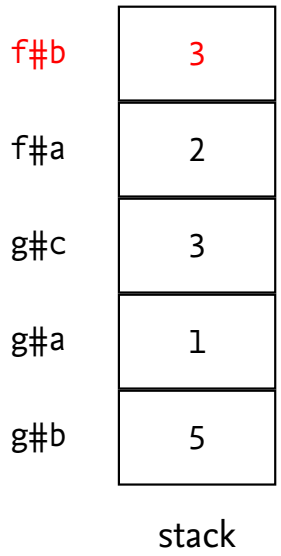


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

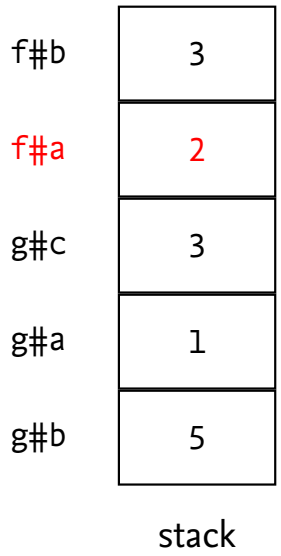


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

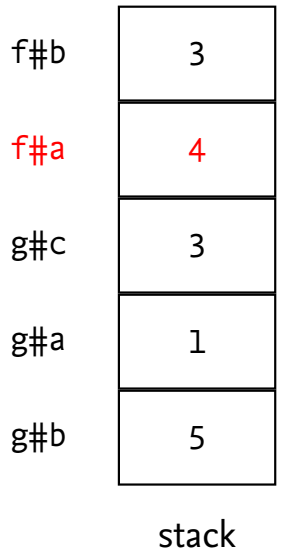


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

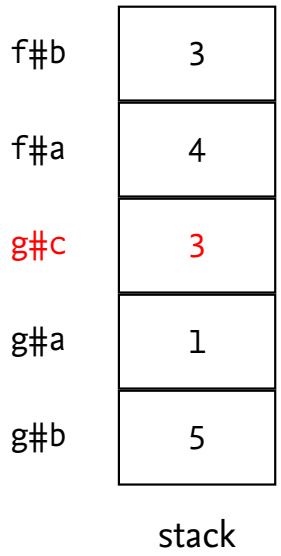


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

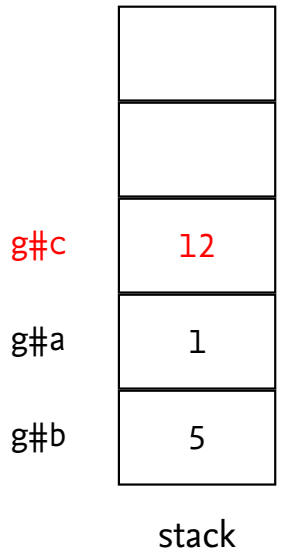


Example #1

Calling g(5)

Java

```
public static int f( int a ) {
    int b = a + 1;
    a = a + 2;
    return a * b;
}

public static void g( int b ) {
    int a = 1;
    int c = 3;

    c = f( a + a );
    System.out.println( a + " " + c );
    // Prints: 1 12
}
```

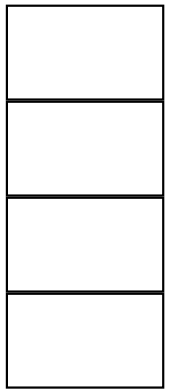


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```



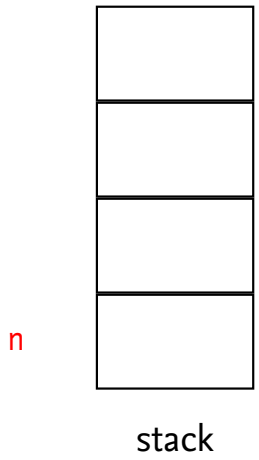
stack

Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

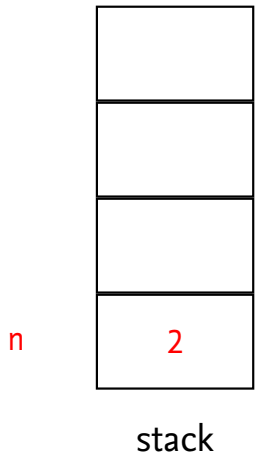


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

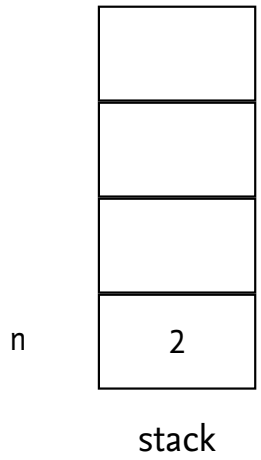


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

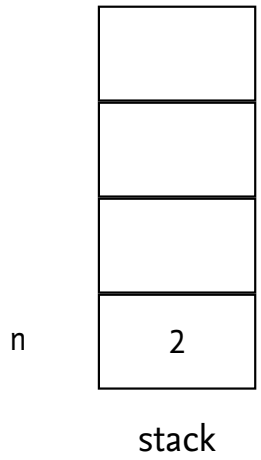


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

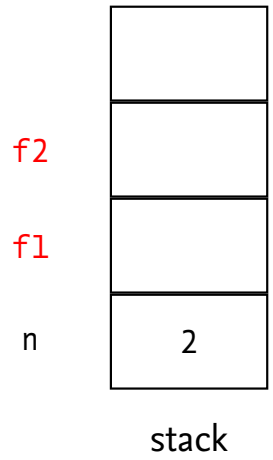


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

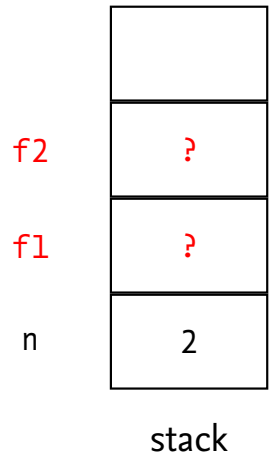


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

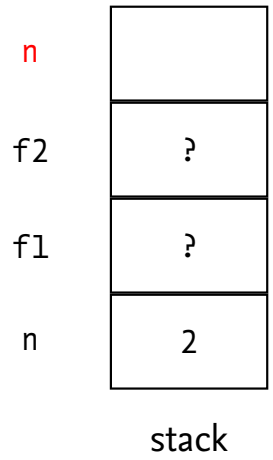


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

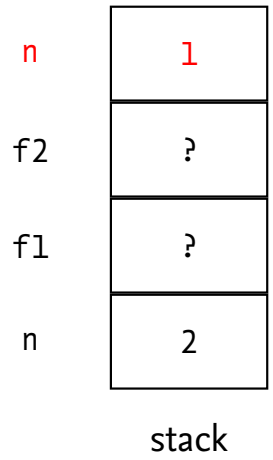


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

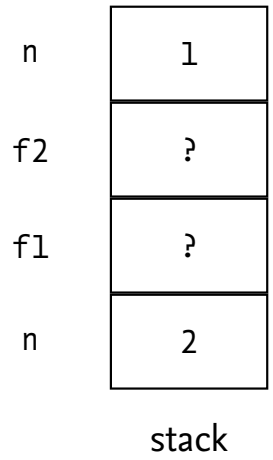


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

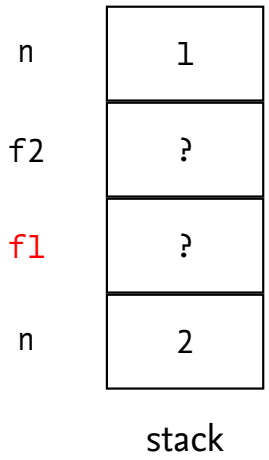


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

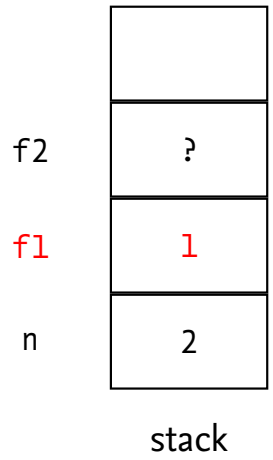


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

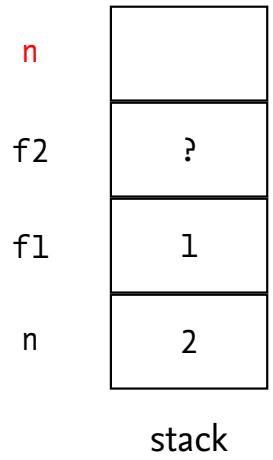


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

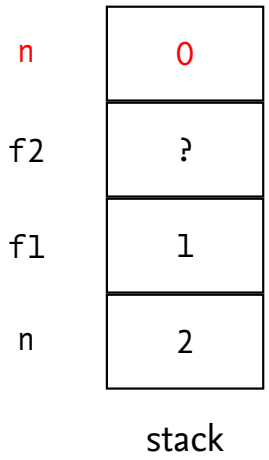


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

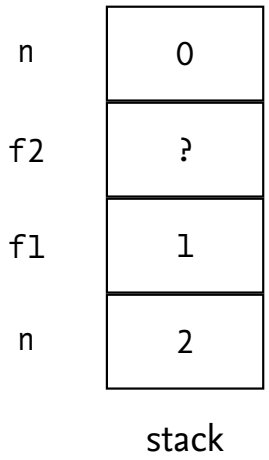


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

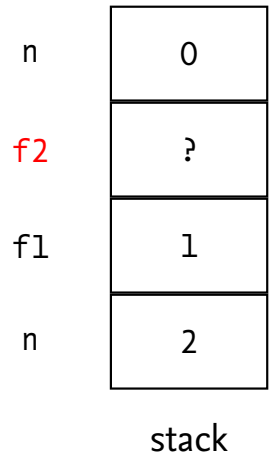


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

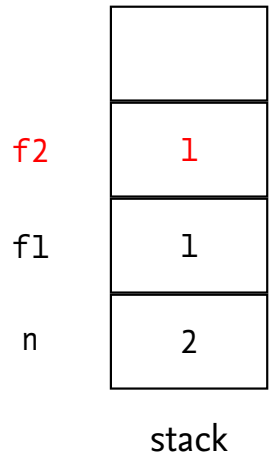


Example #2

Initial Call is `fib( 2 )`

Java

```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```

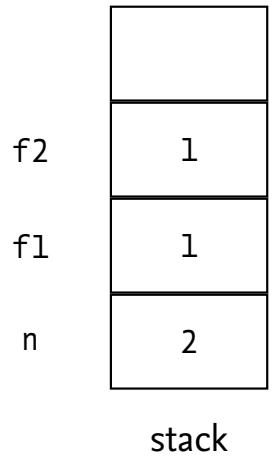


Example #2

Initial Call is `fib( 2 )`

Java

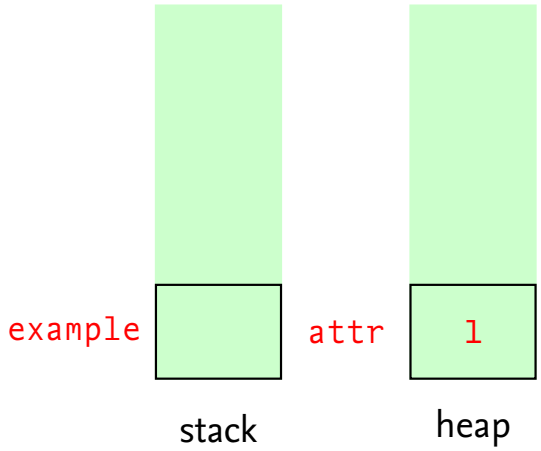
```
public static int fib( int n ) {
    if (n <= 1) {
        return 1;
    } else {
        int f1 = fib( n - 1 );
        int f2 = fib( n - 2 );
        return f1 + f2;
    }
}
```



Example #3: JVM Creates an Object

Java

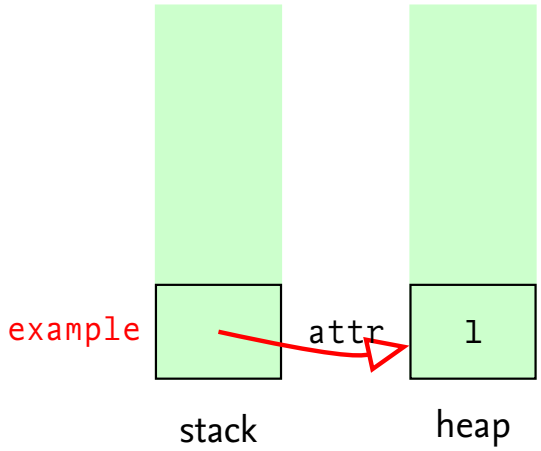
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3: JVM Returns an Object Reference

Java

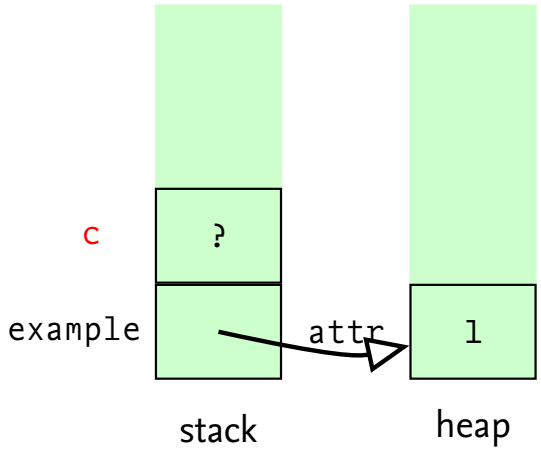
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

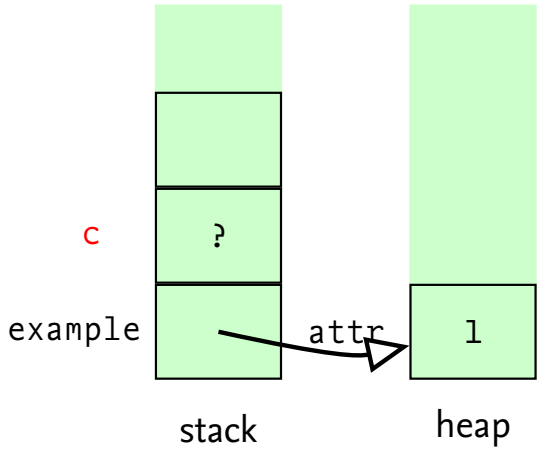
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

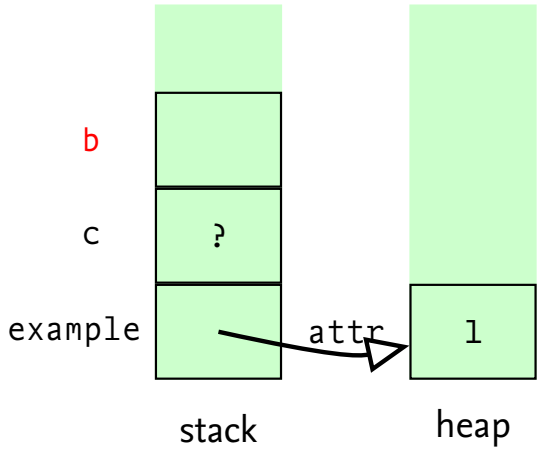
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

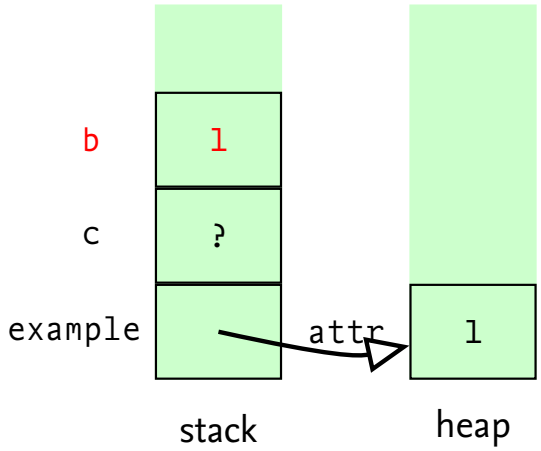
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

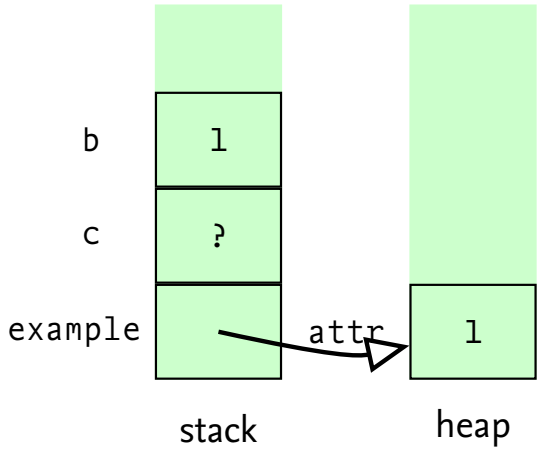
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

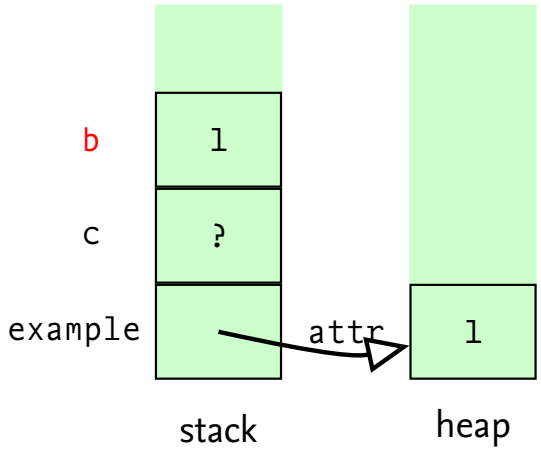
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

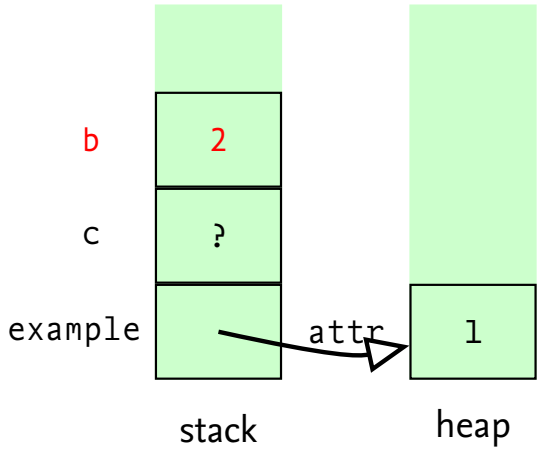
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

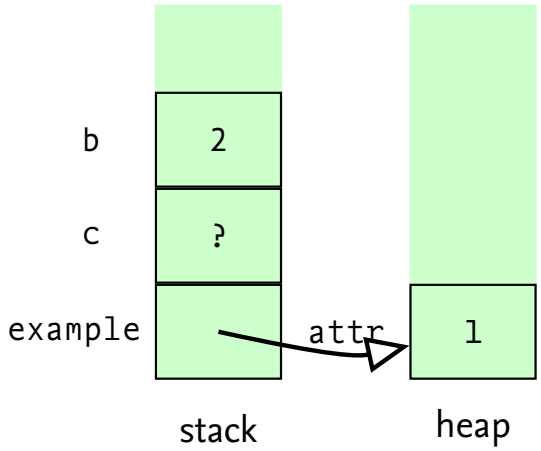
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

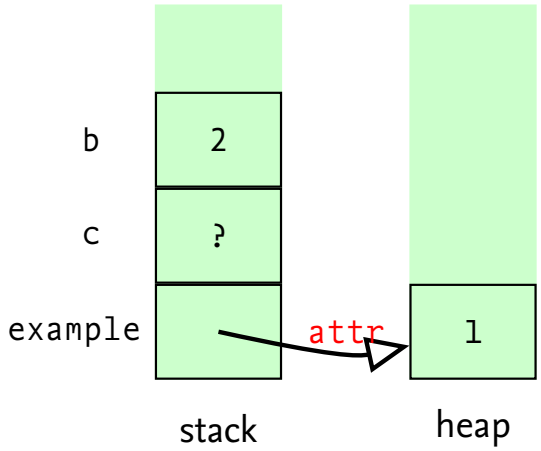
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

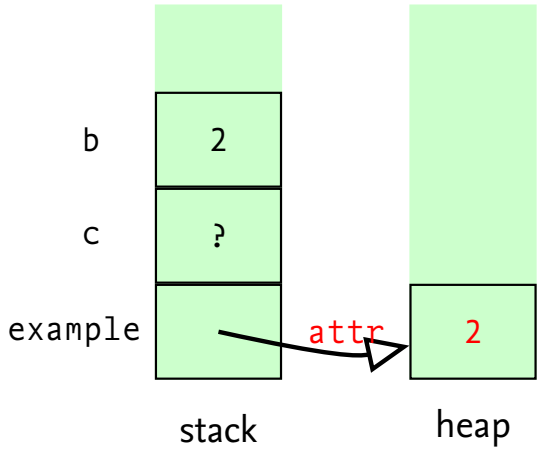
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

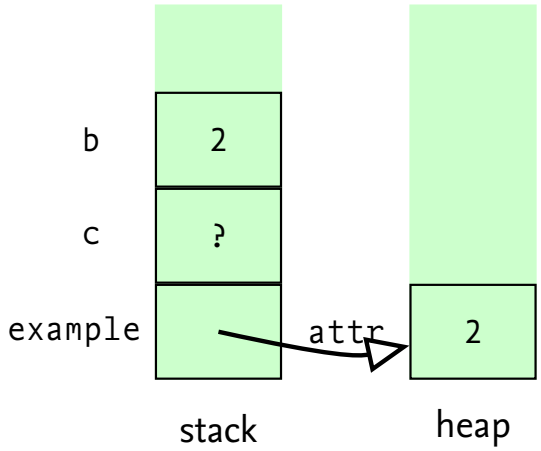
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

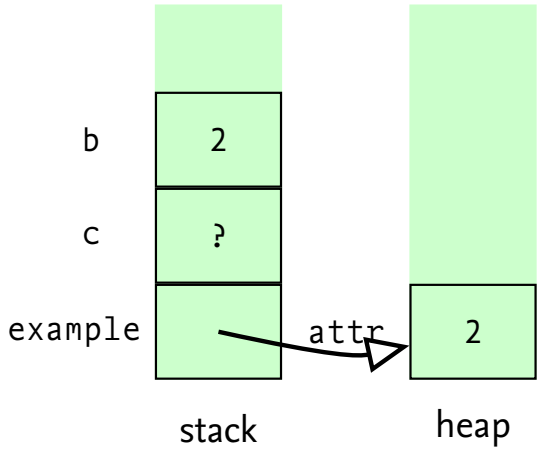
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

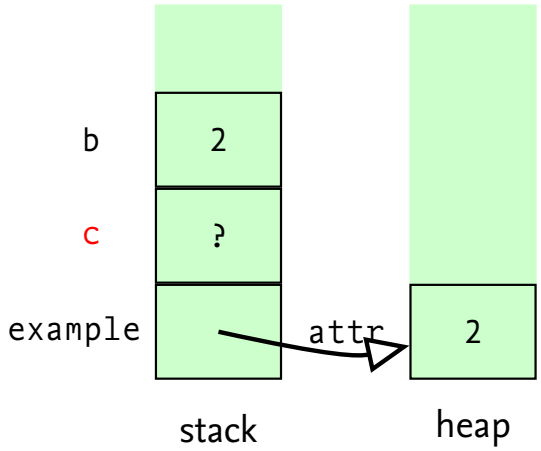
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

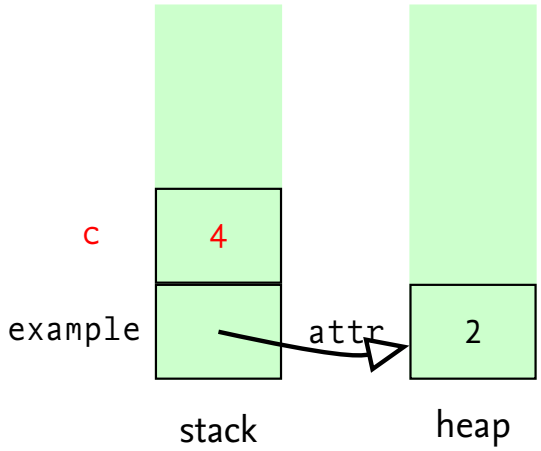
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

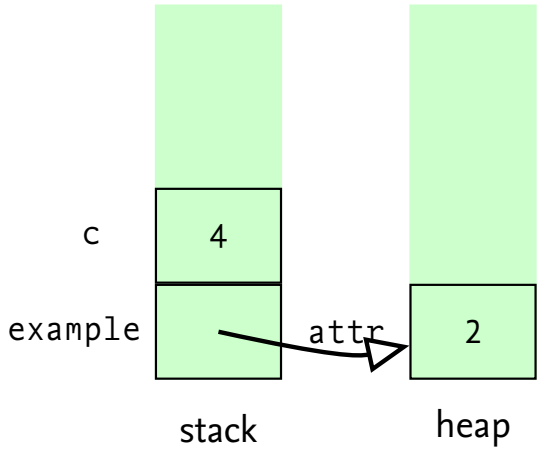
```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );

   }
}
```



Example #3:

Java

```
public class Example {
   private int attr;

   public Example( ) {
       attr = 1;
   }

   public static void main( String[] args ) {
       Example example = new Example( );
       example.g( );
   }

   public int f( int b ) {
      b = 2;
      attr = 2;
      return attr + b;
   }

   public void g( ) {
      int c = f( attr );
      System.out.println( attr + " " + c );
      // Prints: 2 4.
   }
}
```



Class Design: How To?

- When we design an application, how do we choose the classes?
- Once we've decided on the classes,
  - Hhow do we choose the attributes, and
  - Hhow do we choose the methods?
- The answer is in the problem specification.

Clues

- To find classes: look for actors in the spec.
  - This works, because the actors correspond to the objects,
    - And each object is an instance of its class.
  - We may implement the object in a class named after the actor:
    - Toy and `Toy`;
    - Writer and `Writer`;
    - Dog and `Dog`;
    - ...
- The actors do things (verbs): these are the methods.
- The actors own things, these are the attributes.

Clues

- To find classes: look for actors in the spec.
  - This works, because the actors correspond to the objects,
    - And each object is an instance of its class.
  - We may implement the object in a class named after the actor:
    - Toy and `Toy`;
    - Writer and `Writer`;
    - Dog and `Dog`;
    - ...
- The actors do things (verbs): these are the methods.
- The actors own things, these are the attributes.

Clues

- To find classes: look for actors in the spec.
  - This works, because the actors correspond to the objects,
    - And each object is an instance of its class.
  - We may implement the object in a class named after the actor:
    - Toy and `Toy`;
    - Writer and `Writer`;
    - Dog and `Dog`;
    - ...
- The actors do things (verbs): these are the methods.
- The actors own things, these are the attributes.

Playing with Toys

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

Playing with Toys

How do we find the Classes?

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

Playing with Toys

How do we find the Classes? Look for Actors.

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

Playing with Toys

How do we find the Actors?

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

Playing with Toys

How do we find the Actors? Look for Nouns!

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

Playing with Toys

How do we find the Actors? Look for Nouns!

- There are hands and toys;
- Each toy has its own name;
- Each hand has its own type: left or right.
- A toy is (either) used or free;
- Initially, each toy is free;
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, the toy becomes free.

The Toy Class

How do we find the Attributes and Methods?

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Toy Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Toy Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Toy Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Toy Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Toy Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each toy has its own name;
- A toy is (either) used or free;
- Initially, each toy is free;
- When a hand takes a toy, its toy becomes taken;
- When a hand drops its toy, its toy becomes free.

Java

```
public class Toy {
    private final String name;
    private boolean used;

    public Toy( String name ) {
        this.name = name;
        used = false;
    }

    // Getter and setter methods omitted.

    @Override
    public String toString( ) {
        return "Toy[ name = " + name + " ]";
    }
}
```

The Hand Class

How do we find the Attributes and Methods?

- Each hand has its own type: left or right.
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, its toy becomes free.

The Hand Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each hand has its own type: left or right.
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, its toy becomes free.

The Hand Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each hand has its own type: left or right.
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, its toy becomes free.

The Hand Class

How do we find the Attributes and Methods? Look for Properties and (Active) Verbs.

- Each hand has its own type: left or right.
- A hand is (either) empty or full;
- Initially, each hand is empty;
- A hand can only take a free toy;
- A full hand cannot take any toy;
- When a hand takes a toy, the toy becomes taken;
- When a hand takes a toy, the hand becomes full;
- A hand can drop its toy;
- When a hand drops its toy, the hand becomes empty; and
- When a hand drops its toy, its toy becomes free.

The Hand Class

Java

```
public class Hand {
    private final String type;
    private Toy toy;

    public Hand( String type ) {
        this.type = type;
        toy = null;
    }

    public void take( Toy toy ) { ⟨to do⟩ }
    public void drop( ) { ⟨to do⟩ }

    public String getType( ) { return type; }
    public boolean isEmpty( ) { return toy == null; }
    public boolean isFull( ) { return !isEmpty( ); }

    @Override
    public String toString( ) {
        return "Hand[ type = " + type + ", toy = " + toy + " ]";
    }
}
```

The `take( )` Method

Java

```
public void take( Toy toy ) {









}
```

The `take( )` Method

Java

```
public void take( Toy toy ) {

        // We cannot take a Toy if Hand is full.









}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    }









}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    }
        // We cannot take a used Toy.









}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    } else if (toy.getUsed( )) {
        // We cannot take a used Toy.
        System.err.println( "** " + toy + " is taken." );
        System.err.println( "** Cannot take it." );
    }




}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    } else if (toy.getUsed( )) {
        // We cannot take a used Toy.
        System.err.println( "** " + toy + " is taken." );
        System.err.println( "** Cannot take it." );
    } else {
        // Take toy.




    }
}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    } else if (toy.getUsed( )) {
        // We cannot take a used Toy.
        System.err.println( "** " + toy + " is taken." );
        System.err.println( "** Cannot take it." );
    } else {
        // Take toy.
        // Formally mark toy as used.
        toy.setUsed( true );
        // Make toy our current Toy.
        this.toy = toy;
    }
}
```

The take( ) Method

Java

```
public void take( Toy toy ) {
    if (isFull( )) {
        // We cannot take a Toy if Hand is full.
        System.err.println( "** " + this + " is full." );
        System.err.println( "** Cannot take " + toy + "." );
    } else if (toy.getUsed( )) {
        // We cannot take a used Toy.
        System.err.println( "** " + toy + " is taken." );
        System.err.println( "** Cannot take it." );
    } else {
        // Take toy.
        // Formally mark toy as used.
        toy.setUsed( true );
        // Make toy our current Toy.
        this.toy = toy;
    }
}
```

The drop( ) Method

Java

```
public void drop( ) {









}
```

The drop( ) Method

Java

```
public void drop( ) {

        // We can only drop a toy if we have one.









}
```

The drop( ) Method

Java

```
public void drop( ) {
    if (isEmpty( )) {
        // We can only drop a toy if we have one.
        System.err.println( "** " + this + " is empty." );
        System.err.println( "** Cannot drop any toy." );
    }




}
```

The drop( ) Method

Java

```
public void drop( ) {
    if (isEmpty( )) {
        // We can only drop a toy if we have one.
        System.err.println( "** " + this + " is empty." );
        System.err.println( "** Cannot drop any toy." );
    } else {
        // Drop our current toy.




    }
}
```

The drop( ) Method

Java

```
public void drop( ) {
    if (isEmpty( )) {
        // We can only drop a toy if we have one.
        System.err.println( "** " + this + " is empty." );
        System.err.println( "** Cannot drop any toy." );
    } else {
        // Drop our current toy.
        // Formally mark toy as free.
        toy.setUsed( false );
        // Make hand empty.
        toy = null;
    }
}
```

The drop( ) Method

Java

```
public void drop( ) {
    if (isEmpty( )) {
        // We can only drop a toy if we have one.
        System.err.println( "** " + this + " is empty." );
        System.err.println( "** Cannot drop any toy." );
    } else {
        // Drop our current toy.
        // Formally mark toy as free.
        toy.setUsed( false );
        // Make hand empty.
        toy = null;
    }
}
```

The main

Java

```
public static void main( String[] args ) {
    Hand left = new Hand( "left" );
    Hand right = new Hand( "right" );
    Toy game = new Toy( "computer game" );
    Toy puzzle = new Toy( "puzzle" );

    left.take( game );
    right.take( game ); // Results in error message
    right.take( puzzle );
    left.drop( );
    left.drop( );       // Results in error message
}
```

The main: Magic Constants

Java

```
public static void main( String[] args ) {
    Hand left = new Hand( "left" );
    Hand right = new Hand( "right" );
    Toy game = new Toy( "computer game" );
    Toy puzzle = new Toy( "puzzle" );

    left.take( game );
    right.take( game ); // Results in error message
    right.take( puzzle );
    left.drop( );
    left.drop( );       // Results in error message
}
```

The `main`: Constant Class Attributes

Java

```
private static final String LEFT = "left";
private static final String RIGHT = "right";
private static final String GAME = "computer game";
private static final String PUZZLE = "puzzle";

public static void main( String[] args ) {
    Hand left = new Hand( LEFT );
    Hand right = new Hand( RIGHT );
    Toy game = new Toy( GAME );
    Toy puzzle = new Toy( PUZZLE );

    left.take( game );
    right.take( game ); // Results in error message
    right.take( puzzle );
    left.drop( );
    left.drop( );       // Results in error message
}
```

Final Improvement

Java

```
public class Hand {
    public static final String LEFT = "left";
    public static final String RIGHT = "right";
    .
    .
    .
}
```

Java

```
public static void main( String[] args ) {
    Hand left = new Hand( Hand.LEFT );
    Hand right = new Hand( Hand.RIGHT );
    .
    .
    .
}
```

Question Time

Questions Anybody?

For Next Friday

- Study the call-by-value mechanism;
- Carry out the verb and noun analysis on this lecture's example.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.