More Branches
Nested Ifs

Nothing prevents us from nesting if statements:

```java
int i = 2;
if (i > 0) {
    System.out.println("More than 0.");
    if (i > 1) {
        System.out.println("More than 1.");
        if (i > 2) {
            System.out.println("More than 2.");
        }
    }
}
```
**Nested Ifs**

Sometimes nested ifs can be eliminated using logical operators.

```java
if (i > 0) {
    if (i < 3) {
        System.out.println("i is 1 or 2");
    }
}
```

or

```java
if (i > 0 && i < 3) {
    System.out.println("i is 1 or 2");
}
```
TopHat Question 1

What is the output?

```java
boolean a = true;
boolean b = false;
if(a)
    if(b)
        System.out.print("foo");
else
    System.out.print("bar");
```
Dangling Else

Without braces, Java associates the else with the closest if.

```java
boolean a = true;
boolean b = false;
if(a)
    if(b)
        System.out.print("foo");
else
    System.out.print("bar");
```
Dangling Else

Without braces, Java associates the else with the closest if.

```java
boolean a = true;
boolean b = false;
if(a) {
  if(b) {
    System.out.print("foo");
  }
}
else {
  System.out.print("bar");
}
```

Good Programming Practice

Avoid a dangling else; always use braces.
Short-Circuit Evaluation

$expr1 \text{ boolOp } expr2$

- Recall that `&&` and `||` are left to right associative.
- Rule: Only evaluate $expr2$ if necessary.
  - `&&` – if $expr1$ is true.
  - `||` – if $expr1$ is false.

Very useful for avoiding runtime errors. E.g.:

```java
if (b != 0 && a / b > 6)
```
Short-Circuit Evaluation

expr1 boolOp expr2

- Recall that && and || are left to right associative.
- Rule: Only evaluate expr2 if necessary.
  - && – if expr1 is true.
  - || – if expr1 is false.

Very useful for avoiding runtime errors. E.g.:
if (b != 0 && a / b > 6)
public class ShortCircuitEx {
    public static boolean foo(int i) {
        System.out.print("foo");
        if(i > 3) {
            return false;
        }
        return true;
    }

    public static int bar(int i) {
        System.out.print("bar");
        return i % 5;
    }

    public static void main(String[] args) {
        if(foo(5) && bar(5) == 0) {
            System.out.print("--true");
        }
        else {
            System.out.print("--false");
        }
    }
}
Conditional Assignment

\[ \text{someVar} = \text{cond} ? \text{trueExpr} : \text{falseExpr} \]

Ternary Operator ( ? : )

Equivalent to:

```java
if (cond) {
    someVar = trueExpr;
}
else {
    someVar = falseExpr;
}
```
TopHat Question 3

What is the value contained in the String object referenced by s?

```java
int i = 4;
String s = (i % 2 == 0) ? "i is odd" : "i is even";
```
More Primitives
## Integer Overflow

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>–128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>–32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>–2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
</tbody>
</table>

### Overflow Error

- Storing numbers that require more bits than available.
- No runtime error; results not as expected.
- **Fixes:**
  - Use wider type
  - Restrict input
  - Adjust calculations
TopHat Question 4

What is the output?

double d1 = 3.6;
double d2 = 3.6;
double d3 = 1.2 * 3.0;
if(d1 == d2) System.out.println("Eq1");
if(d1 == d3) System.out.println("Eq2");
Floating-point Comparison

Comparison with ==

- Don’t use == to compare float or double.
- Floating-point rounding errors make this unreliable.
Floating-point Comparison

Comparison with ==
- Don’t use == to compare float or double.
- Floating-point rounding errors make this unreliable.

ε-Close Comparison

Math.abs(someFloat1 - someFloat2) < someEps

Two floating-point types are considered equal if they are within an ε (small value) of each other.
More Methods
Scope

### Code Block in Java

A *code block* is any number of statements between a pair of braces.

### Variable Scope

- The *scope* of a variable is the region of code where it can be used.
- A variable’s scope is limited to the code block in which it is declared.
- A variable can only be used after it is declared.
- A local variable cannot be redeclared in nested code block.
Scope

```java
public class ScopeEx {

    public static void someMethod(int p) {
        // instructions with p
        int i;
        // instructions with p and i
        // instructions with j, k and l not authorised
        int j;
        // instructions with p, i and j authorised
        // instructions with k and l not authorised
        {
            // instructions with p, i and j authorised
            // instructions with k and l not authorised
            int k;
            // instructions with p, i, j and k authorised
            // instructions with l not authorised
        }
        // instructions with p, i and j authorised
        // instructions with k and l not authorised
        int l;
        // instructions with p, i, j and l authorised
        // instructions with k not authorised
    }

    public static void anotherMethod() {
        // No variables in scope.
        i = 5;  // Not authorised; i is out of scope.
        p = 6;  // Not authorised; i is out of scope.
    }
}
```
Overloading Methods

Recall: Method Signature

A method signature consists of the method name and the types of the parameters.
**Overloading Methods**

**Recall: Method Signature**

A *method signature* consists of the method name and the types of the parameters.

**Method Overloading**

- Multiple methods with different signatures, but the same name.
- When an overloaded method is called, Java uses the argument types to determine which method to use.
TopHAT Question 5

What is the output?

```java
public class OverloadEx {

    public static int overload() {
        return 0;
    }

    public static int overload(String s) {
        return s.length();
    }

    public static String overload(int i) {
        return "i = " + i;
    }

    public static void main(String[] args) {
        System.out.print(overload("Test"));
    }
}
```
TopHat Question 6

What is the output?

```java
public class CompositionEx {

    public static int f(int i) {
        return i * i;
    }

    public static int g(int i) {
        return i % 2 == 0 ? i / 2 : i;
    }

    public static int h(int i) {
        return i + 3;
    }

    public static void main(String[] args) {
        System.out.print(f(g(h(5))));
    }
}
```
**More Method Calling**

**Composition**

```java
public class CompositionEx {

    public static int f(int i) {
        return i * i;
    }

    public static int g(int i) {
        return i % 2 == 0 ? i / 2 : i;
    }

    public static int h(int i) {
        return i + 3;
    }

    public static void main(String[] args) {
        int i = h(5);
        i = g(i);
        System.out.print(f(i));
    }
}
```
More Method Calling

Methods Calling Methods

public class MetCallMetEx {

    public static int f(int i) {
        return i * i;
    }

    public static int g(int i) {
        return f(i);
    }

    public static int h(int i) {
        int j = f(i);
        return g(j);
    }

    public static void main(String[] args) {
        System.out.print(h(2));
    }
}

More Objects (and Methods)
Objects and NULL

null

- "Nothing" reference.
Objects and Null

null

- “Nothing” reference.
- Any reference type can be set to null.
Objects and Null

null

- “Nothing” reference.
- Any reference type can be set to null.
- In Java, null equals null is true.
  ```java
  String s = null;
  boolean b = s == null; //True
  ```
Objects and Null

null

- “Nothing” reference.
- Any reference type can be set to null.
- In Java, null equals null is true.
  ```java
  String s = null;
  boolean b = s == null; // True
  ```

Empty String vs null

- Empty string "": is a string object that is empty.
  ```java
  String s = ""; // Reference to ""
  ```
- null: is not an instance of string – no reference.
  ```java
  String t = null; // No reference
  ```
TopHat Question 7

What is the output?

String s = null;
System.out.print(s.contains("foo"));

a. Nothing – Runtime error
b. false
c. true
More Memory Model

Stack

Heap

Recall: Memory Address

Every byte in memory has a unique memory address.
More Memory Model

Stack

The Stack
- Local variables
- Automatically managed
- Used sequentially

Heap
More Memory Model

### Stack
- Local variables
- Automatically managed
- Used sequentially

### Heap
- Also called *free memory*.
- Region where objects are allocated memory.
- Memory management is done by the JVM’s *garbage collection*.
- Allocations are arbitrary.

Recall: Memory Address

Every byte in memory has a unique memory address.
More Memory Model

Stack

- Local variables
- Automatically managed
- Used sequentially

Heap

- Also called free memory.
- Region where objects are allocated memory.
- Memory management is done by the JVM’s garbage collection.
- Allocations are arbitrary.

Recall: Memory Address

Every byte in memory has a unique memory address.
A Day in the Life of the Stack

Stack

```
1 public class StackHeapEx {
2
3     public static int g(int i) {
4         return i * i;
5     }
6
7     public static int f(int i) {
8         int j = g(i);
9         return i + j;
10     }
11
12     public static void main(String[] args) {
13         int k = 4;
14         k = f(k);
15     }
16 }
```
A Day in the Life of the Stack

Stack

```
public class StackHeapEx {

    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

Stack

```
public class StackHeapEx {

    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

public class StackHeapEx {

    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
A Day in the Life of the Stack

```java
public class StackHeapEx {

    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }
    public static int f(int i) {
        int j = g(i);
        return i + j;
    }
    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

Stack

1 public class StackHeapEx {
2     
3         public static int g(int i) {
4             return i * i;
5         }
6     
7         public static int f(int i) {
8             int j = g(i);
9             return i + j;
10        }
11     
12         public static
13         void main(String[] args) {
14             int k = 4;
15             k = f(k);
16         }
17     
}
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }
    public static int f(int i) {
        int j = g(i);
        return i + j;
    }
    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }
    public static int f(int i) {
        int j = g(i);
        return i + j;
    }
    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

Stack

Stack Frames

main

1 public class StackHeapEx {
2     
3         public static int g(int i) {
4             return i * i;
5         }
6
7         public static int f(int i) {
8             int j = g(i);
9             return i + j;
10         }
11
12         public static void main(String[] args) {
13             int k = 4;
14             k = f(k);
15         }
16     
17 }
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }
    public static int f(int i) {
        int j = g(i);
        return i + j;
    }
    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }
    public static int f(int i) {
        int j = g(i);
        return i + j;
    }
    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

```java
public class StackHeapEx {
    public static int g(int i) {
        return i * i;
    }

    public static int f(int i) {
        int j = g(i);
        return i + j;
    }

    public static void main(String[] args) {
        int k = 4;
        k = f(k);
    }
}
```
A Day in the Life of the Stack

Stack

Stack Frames

main

20

stack

k

1  public class StackHeapEx {
2                                              
3  public static int g(int i) {
4       return i * i;
5  }
6                                              
7  public static int f(int i) {
8       int j = g(i);
9       return i + j;
10      }
11                                              
12  public static
13  void main(String[] args) {
14       int k = 4;
15       k = f(k);
16  }
17  }
**Garbage Collection**

**Cleaning up the Heap**

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the *garbage collector*.

![Diagram of Stack and Heap with objects and references](image-url)
Garbage Collection

Cleaning up the Heap

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the garbage collector.
GARbage COLleCtion

Cleaning up the Heap

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the garbage collector.

Stack

Heap

Objects in Heap:
- "foob"ar"
- "foo"

Objects in Stack:
- j
- str

Garbage Collector
**Garbage Collection**

**Cleaning up the Heap**

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the *garbage collector*.

![Diagram showing stack and heap with garbage collection](image)
GARbage Collection

Cleaning up the Heap

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the garbage collector.
GARBAGE COLLECTION

Cleaning up the Heap

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the garbage collector.

Stack

Heap

\[ \text{j} \]
\[ \text{str} \]

"foo"

5

6
**Garbage Collection**

**Cleaning up the Heap**

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the *garbage collector*.

Stack

Heap

```
  5

"foo"
```

```
j
str
```
Garbage Collection

Cleaning up the Heap

- JVM periodically checks the objects on the heap.
- Objects with no references are removed by the garbage collector.
FURTHER READING

COMP SCI 200: Programming I
zyBook code:
WISCCOMPSCI200Fall2017

- Chapter 5. More Primitives, Objects, Branches & Methods
References
Image Sources I

http://www.p2pays.org/socialmarketing/pilotproject.asp

https://brand.wisc.edu/web/logos/

http://www.zybooks.com/