Boolean Statements
**Booleans**

**Primitive**

- `boolean b;`
- Values: true or false
- Wrapper class: `Boolean`
**Boolean Expressions**

An expression that evaluates to a boolean. I.e. true or false.
**Boolean Expressions**

An expression that evaluates to a boolean. I.e. true or false.

**Comparison Operators**
- Binary \((expr1 \text{ oper } expr2)\):
  - ==: Equal to
  - !=: Not equal to
  - >: Greater than
  - <: Less than
  - >=: Greater than or equal
  - <=: Less than or equal
Boolean Expressions

An expression that evaluates to a boolean. I.e. true or false.

Logical Operators

- Unary \((\text{oper expr})\):
  - \(!\) : Logical NOT
- Binary \((\text{expr1 oper expr2})\):
  - \(\&\&\) : Logical AND
  - \(||\) : Logical OR
- Don’t confuse with the \textit{bitwise} operators: &,, |,, and ^.
## Operator Precedence

**Operator Precedence Table:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>higher</td>
<td>( &lt;expression&gt; )</td>
<td>grouping with parentheses</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>[ ] ( ) .</td>
<td>array index, method call, member access (dot operator)</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>++ --</td>
<td>post-increment, post-decrement</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>++ ++ + -- !</td>
<td>pre-increment, unary plus/minus, logical negation</td>
<td>right to left</td>
</tr>
<tr>
<td></td>
<td>(type) new</td>
<td>casting and creating object</td>
<td>right to left</td>
</tr>
<tr>
<td></td>
<td>* / %</td>
<td>multiplication, division, modulus</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>+ - +</td>
<td>addition, subtraction, concatenation</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>relational and Java’s instanceof operator</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>instanceof</td>
<td>equality</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>== !=</td>
<td>conditional AND (short-circuits)</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
<td>conditional OR (short-circuits)</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>? :</td>
<td>assignment</td>
<td>right to left</td>
</tr>
<tr>
<td>lower</td>
<td>= += -= *= /= %==</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Truth Tables**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a ∧ b</th>
<th>a ∨ b</th>
<th>a ⊕ b</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Logic Rules**

- **AND (&&, &)** – True only if both operands are true.
- **OR (||, |)** – True if either operand is true.
- **XOR (ˆ)** – *Exclusive OR*; True if one (but not both) operand(s) is true.
- **NOT (!)** – Flips the truth value.
## Truth Tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a ∧ b</th>
<th>a ∨ b</th>
<th>a ⊕ b</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Logic Rules

- **AND** (&&, &) – True only if both operands are true.
- **OR** (||, |) – True if either operand is true.
- **XOR** (^) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- **NOT** (!) – Flips the truth value.
**Truth Tables**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logic Rules**

- **∧** AND (&&, &) – True only if both operands are true.
- **∨** OR (||, |) – True if either operand is true.
- **⊕** XOR (ˆ) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- **¬** NOT (!) – Flips the truth value.
## Truth Tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a \land b$</th>
<th>$a \lor b$</th>
<th>$a \oplus b$</th>
<th>$\neg a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Tables

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a ∧ b</td>
<td>a ∨ b</td>
<td>a ⊕ b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Logic Rules

∧  AND (&&, &) – True only if both operands are true.
∨  OR (||, |) – True if either operand is true.
⊕  XOR (^) – Exclusive OR; True if one (but not both) operand(s) is true.
¬  NOT (!) – Flips the truth value.
Truth Tables

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a ∧ b</td>
<td>a ∨ b</td>
<td>a ⊕ b</td>
<td>¬a</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Logic Rules

∧ AND (&&, &) – True only if both operands are true.

∨ OR (||, |) – True if either operand is true.

⊕ XOR (^) – Exclusive OR; True if one (but not both) operand(s) is true.

¬ NOT (!) – Flips the truth value.
**Truth Tables**

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a ∧ b</th>
<th>a ∨ b</th>
<th>a ⊕ b</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logic Rules**

- **AND (&&, &)** – True only if both operands are true.
- **OR (||, |)** – True if either operand is true.
- **XOR (ˆ)** – Exclusive OR; True if one (but not both) operand(s) is true.
- **NOT (!)** – Flips the truth value.
## Truth Tables

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a \land b$</th>
<th>$a \lor b$</th>
<th>$a \oplus b$</th>
<th>$\neg a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Logic Rules

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
<td>AND ($&amp;&amp;$, $&amp;$) – True only if both operands are true.</td>
</tr>
<tr>
<td>$\lor$</td>
<td>OR ($</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>XOR ($^\wedge$) – Exclusive OR; True if one (but not both) operand(s) is true.</td>
</tr>
<tr>
<td>$\neg$</td>
<td>NOT ($!$) – Flips the truth value.</td>
</tr>
</tbody>
</table>
**Truth Tables**

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a \wedge b$</th>
<th>$a \vee b$</th>
<th>$a \oplus b$</th>
<th>$\neg a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logic Rules**

$\wedge$ AND ($\&\&$, $\&$) – True only if both operands are true.

$\vee$ OR ($||$, $|$) – True if either operand is true.

$\oplus$ XOR ($\hat{\vee}$) – Exclusive OR; True if one (but not both) operand(s) is true.

$\neg$ NOT ($!$) – Flips the truth value.
Truth Tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a ∧ b</th>
<th>a ∨ b</th>
<th>a ⊕ b</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logic Rules

∧  AND (&&, &) – True only if both operands are true.
∨  OR (||, |) – True if either operand is true.
⊕  XOR (^) – Exclusive OR; True if one (but not both) operand(s) is true.
¬  NOT (!) – Flips the truth value.
## Truth Tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(a \land b)</th>
<th>(a \lor b)</th>
<th>(a \oplus b)</th>
<th>(\neg a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Logic Rules

- \(\land\) AND (&&, &) – True only if both operands are true.
- \(\lor\) OR (||, |) – True if either operand is true.
- \(\oplus\) XOR (ˆ) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- \(\neg\) NOT (!) – Flips the truth value.
Truth Tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( a \land b )</th>
<th>( a \lor b )</th>
<th>( a \oplus b )</th>
<th>( \neg a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logic Rules

- **\( \land \)**: AND (&&, &) – True only if both operands are true.
- **\( \lor \)**: OR (||, |) – True if either operand is true.
- **\( \oplus \)**: XOR (ˆ) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- **\( \neg \)**: NOT (!) – Flips the truth value.
Truth Tables

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a ∧ b</td>
<td>a ∨ b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logic Rules

∧  AND (&&, &) – True only if both operands are true.

∨  OR (||, |) – True if either operand is true.

⊕  XOR (^) – *Exclusive OR*; True if one (but not both) operand(s) is true.

¬  NOT (!) – Flips the truth value.
**Truth Tables**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a ∧ b</td>
<td>a ∨ b</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Logic Rules**

- **∧** AND (`&`, `&&`) – True only if both operands are true.
- **∨** OR (`||`, `|`) – True if either operand is true.
- **⊕** XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- **¬** NOT (`!`) – Flips the truth value.
Truth Tables

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a (\land) b</td>
<td>a (\lor) b</td>
<td>a (\oplus) b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Logic Rules

\(\land\) AND (\&\&, \&) – True only if both operands are true.

\(\lor\) OR (\|\|, |) – True if either operand is true.

\(\oplus\) XOR (^) – *Exclusive OR*; True if one (but not both) operand(s) is true.

\(\neg\) NOT (!) – Flips the truth value.
Truth Tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a ∧ b</th>
<th>a ∨ b</th>
<th>a ⊕ b</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Logic Rules

∧  AND (&&, &) – True only if both operands are true.

∨  OR (||, |) – True if either operand is true.

⊕  XOR (^) – Exclusive OR; True if one (but not both) operand(s) is true.

¬  NOT (!) – Flips the truth value.
# Truth Tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(a \land b)</th>
<th>(a \lor b)</th>
<th>(a \oplus b)</th>
<th>(\neg a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Logic Rules**

- \(\land\) **AND** (&&, &) – True only if both operands are true.
- \(\lor\) **OR** (||, |) – True if either operand is true.
- \(\oplus\) **XOR** (^) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- \(\neg\) **NOT** (!) – Flips the truth value.
TopHat Question 1

If A is 1 and B is 1, what is \(!A \ || \ B\)?

a. 1
b. 0
**Bitwise Operations**

Performs the logical operations on each of the corresponding bits of the operands.

---

**Bitwise Operators**

- Binary \((expr1 \text{ oper } expr2)\):
  - & – Bitwise AND
  - | – Bitwise OR
  - ^ – Bitwise XOR
**Bitwise Operations**

Performs the logical operations on each of the corresponding bits of the operands.

E.g. $7 \& 5 = 5$

Consider the bits:

```
111
& 101
-----
101
```

**Bitwise Operators**

- Binary $(expr1 \ oper \ expr2)$:
  - $\&$ – Bitwise AND
  - $|$ – Bitwise OR
  - $^\wedge$ – Bitwise XOR
**TopHat Question 2**

What is $7 \mid 5$?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>12</td>
</tr>
<tr>
<td>b.</td>
<td>7</td>
</tr>
<tr>
<td>c.</td>
<td>5</td>
</tr>
<tr>
<td>d.</td>
<td>3</td>
</tr>
</tbody>
</table>
Comparing References
Reference Comparison

Equality Operator (==)

- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.
Reference Comparison

Equality Operator (==)
- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.

```java
Integer i = 6;
Integer j = 6;
Integer k = new Integer(6);
```
**Reference Comparison**

**Equality Operator (==) [Resp: Inequality Operator (!=)]**

- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.

```
Integer i = 6;
Integer j = 6;
Integer k = new Integer(6);
```
Comparing Values of Primitive Wrappers

```java
Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;
```

_instance Methods for Comparison_

- `equals(otherObj)` – True if values of the objects are equal. False otherwise.
  
  E.g. `i.equals(k) // true`
Comparing Values of Primitive Wrappers

Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;

Instance Methods for Comparison

- equals(otherObj) – True if values of the objects are equal. False otherwise.
  E.g. i.equals(k) // true

- compareTo(otherObj) – 0 if equal, negative if less than otherObj, and positive if greater than otherObj.
  E.g. i.compareTo(j) // +ve
Comparing Values of Primitive Wrappers

```java
Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;
```

Instance Methods for Comparison

- `equals(otherObj)` – True if values of the objects are equal. False otherwise.
  
  E.g. `i.equals(k) // true`

- `compareTo(otherObj)` – 0 if equal, negative if less than `otherObj`, and positive if greater than `otherObj`.
  
  E.g. `i.compareTo(j) // +ve`

Other Comparison Operators

- (In)Equality exception: `j == m // true`

- Other operators work as expected: `<, <=, >, >=.`
String Comparison

Only Instance Methods

- equals(otherStr)
- compareTo(otherStr)

Lexicographical Comparison
Starting from index 0 and compare each pair of characters at index \( x \) according to the following rules:

1. The character with the smaller Unicode value is considered smaller lexicographically.
2. No character is smaller than an existing character. (E.g. "foo" is smaller than "foobar")
**String Comparison**

**Only Instance Methods**
- `equals(otherStr)`
- `compareTo(otherStr)`

**Lexicographical Comparison**
Starting from index 0 and compare each pair of characters at index `x` according to the following rules:

1. The character with the smaller Unicode value is considered smaller lexicographically.
2. No character is smaller than an existing character. (E.g. "foo" is smaller than "foobar")
**String Comparison**

### Only Instance Methods

- `equals(otherStr)`
- `compareTo(otherStr)`
- `equalsIgnoreCase(otherStr)`
- `compareToIgnoreCase(otherStr)`

### Lexicographical Comparison

Starting from index 0 and compare each pair of characters at index $x$ according to the following rules:

1. The character with the smaller Unicode value is considered smaller lexicographically.
2. No character is smaller than an existing character. (E.g. "foo" is smaller than "foobar")
TopHat Question 3

Arrange these words in lexicographical order (smallest to largest):

a. Javascript
b. C
c. Java
d. C++
e. C#
Conditional Statements
**If Statement**

```java
if (cond) {
    trueStatement;
    .
    .
    .
    lastTrueStatement;
}
```
If Statement

```java
if (cond) {
    trueStatement;
    .
    .
    .
    lastTrueStatement;
}
```
If-Else Statement

```java
if (cond) {
    trueStatement;
    ...
    lastTrueStatement;
}
else {
    falseStatement;
    ...
    lastFalseStatement;
}
```
**If-Else Statement**

```java
if (cond) {
    trueStatement;
    ...
    lastTrueStatement;
}
else {
    falseStatement;
    ...
    lastFalseStatement;
}
```

![Control Flow Diagram]

- **cond**
- **true**
- **false**
- **True Stmt Block**
- **False Stmt Block**
- **Following statements**
TopHat Question 4

What is printed out in the following code block:

```java
boolean toBe = false;
if(toBe || !toBe) {
    System.out.print("Tautology");
}
else {
    System.out.print("Contradiction");
}
```
TopHat Question 5

What is the boolean expression to complete the following code?

Assume that i is an int variable. Replace ?????? in the code with the appropriate boolean expression.

```java
if( ?????? ) {
    System.out.println(i + " is divisible by 3");
} else {
    System.out.println(i + " is not divisible by 3");
}
```
**If-Else If Statement**

**Ending Else**

```java
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.
else {
    allFalseStatements;
}
```
**If-Else If Statement**

**Ending Else**

```java
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}

else {  
    allFalseStatements;
}
```

Control Flow Diagram:
- True 1 Stmt Block: True 1
- True 2 Stmt Block: True 2
- Else Stmt Block: False
- Following statements: Always executes

Decision Nodes:
- cond1: True/False
- cond2: True/False

Flow:
- True 1 to cond1
- True 2 to cond2
- False to Else Stmt
- All False to Else Stmt
- Following statements to Else Stmt
If-Else If Statement
No Ending Else

```java
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}

else if (condN) {
    trueStatementsN;
}
```

Control Flow

<table>
<thead>
<tr>
<th>cond1</th>
<th>True 1</th>
<th>Stmt Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond2</td>
<td>True 2</td>
<td>Stmt Block</td>
</tr>
<tr>
<td>condN</td>
<td>True N</td>
<td>Stmt Block</td>
</tr>
</tbody>
</table>

Following statements:

<table>
<thead>
<tr>
<th>true</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
If-Else If Statement
No Ending Else

```java
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.
else if (condN) {
    trueStatementsN;
}
```

Control Flow Diagram:
**Multiple if**

```java
if (cond1) {
    trueStatements1;
}
if (cond2) {
    trueStatements2;
}
...
if (condN) {
    trueStatementsN;
}
```
Multiple if statements

```java
if (cond1) {
    trueStatements1;
}
if (cond2) {
    trueStatements2;
}
.
.
.
if (condN) {
    trueStatementsN;
}
```

Control Flow Diagram:
- True 1 Stmt Block
- True 2 Stmt Block
- True N Stmt Block
- Following statements
TopHat Question 6

What does the following code print?

```java
int v1 = 3, v2 = 6;
if (v1 + 2 < v2) System.out.print(" tic ");
else if (v1 + 4 < v2) System.out.print(" tac ");
else if (v1 + 4 > v2) System.out.print(" toe ");
```
What does the following code print?

```java
int v1 = 3, v2 = 6;
if (v1 + 2 < v2) System.out.print(" tic ");
if (v1 + 4 < v2) System.out.print(" tac ");
if (v1 + 4 > v2) System.out.print(" toe ");
```
Switch Statements

```java
switch (expr) {
    case const1:
        case1Statements;
        break;

    case const2:
        case2Statements;
        break;

    ...

    default:
        defStatements;
        break;
}
```
Switch Statements

```java
switch (expr) {
    case const1:
        case1Statements;
        break;
    case const2:
        case2Statements;
        break;
    .
    .
    .
    default:
        defStatements;
        break;
}
```
Switch Statements
Control flow depends on break;

```java
switch (expr) {
    case const1:
        case1Statements;
        break;

    case const2:
        case2Statements;
        break;

    default:
        defStatements;
        break;
}
```

Control Flow

- `expr == const1` (true)
- `expr == const2` (true)
- No match

Diagram:
- `expr == const1`: Case 1 Stmt Block
- `expr == const2`: Case 2 Stmt Block
- Default Block
- Following statements
**Switch Statements**

Control flow depends on `break;`

```java
switch (expr) {
    case const1:
        case1Statements;
    case const2:
        case2Statements;
    break;
    ...
    default:
        defStatements;
        break;
}
```
Switch Statements Keys

Rules

- Finds the *first* match going from the top to the bottom in order.
- *Fall through* – A case without a break will fall through to the next case’s statements.
Switch Statements Keys

Rules
- Finds the *first* match going from the top to the bottom in order.
- *Fall through* – A case without a *break* will fall through to the next case’s statements.

Good Practices
- Only use fall though when the cases execute the same code.
- Generally, it is best to always have a *default* case at the end.
If-Else if and Switch Statements

```java
if (i == 1) {
    System.out.println("one");
}
else if (i == 2) {
    System.out.println("two");
}
else {
    System.out.println("Not 1 or 2");
}
```
If-Else if and Switch Statements

    if (i == 1) {
        System.out.println("one");
    }
    else if (i == 2) {
        System.out.println("two");
    }
    else {
        System.out.println("Not 1 or 2");
    }

    switch (i) {
    case 1:
        System.out.println("one");
        break;
    case 2:
        System.out.println("two");
        break;
    default:
        System.out.println("Not 1 or 2");
        break;
    }
TopHat Question 8

What does the following code print?

```java
int month = 8;
switch (month) {
    case 1: System.out.println("January");
            break;
    case 2: System.out.println("February");
            break;
    case 3: System.out.println("March");
            break;
    case 4: System.out.println("April");
            break;
    case 5: System.out.println("May");
            break;
    case 6: System.out.println("June");
            break;
    case 7: System.out.println("July");
            break;
    case 8: System.out.println("August");
            break;
    case 9: System.out.println("September");
            break;
    case 10: System.out.println("October");
             break;
    case 11: System.out.println("November");
             break;
    case 12: System.out.println("December");
             break;
    default: System.out.println("Invalid month");
             break;
}
```
What does the following code print?

```java
int month = 8;
switch (month) {
    case 1: System.out.println("January");
    case 2: System.out.println("February");
    case 3: System.out.println("March");
    case 4: System.out.println("April");
    case 5: System.out.println("May");
    case 6: System.out.println("June");
    case 7: System.out.println("July");
    case 8: System.out.println("August");
    case 9: System.out.println("September");
    case 10: System.out.println("October");
    case 11: System.out.println("November");
    case 12: System.out.println("December");
        break;
    default: System.out.println("Invalid month");
        break;
}
```
Write a method that takes an integer as input and returns a string with the integer converted to an abbreviated ordinal. I.e. 1 becomes 1st, 2 becomes 2nd, 3 becomes 3rd, etc...
Using Eclipse
**ECLIPSE**

**Eclipse**

- Integrated development environment (IDE) that we will use in the course.
- In 2017, 40.5% use Eclipse (48% use IntelliJ) \(^a\)
- In 2016, 48.2% use Eclipse (43.6% use IntelliJ) \(^b\)

\(^a\)Source: http://www.baeldung.com/java-in-2017
\(^b\)Source: http://www.baeldung.com/java-ides-2016
Eclipse

- Integrated development environment (IDE) that we will use in the course.
- In 2017, 40.5% use Eclipse (48% use IntelliJ) \(^a\)
- In 2016, 48.2% use Eclipse (43.6% use IntelliJ) \(^b\)

\(^b\) Source: [http://www.baeldung.com/java-ides-2016](http://www.baeldung.com/java-ides-2016)

Installing and Using

Style Guide


Examples:

- http://pages.cs.wisc.edu/~cs200/resources/Frame.java
- http://pages.cs.wisc.edu/~cs200/resources/Circle.java

Key Elements

- File Comment Header
- Class Comment Header
- Method Comment Header
- Good names with proper formatting
- Proper use of whitespace
FURTHER READING

COMP SCI 200: Programming I
zyBook code:
WISCCOMPSCI200Fall2017

- Chapter 3. Using Objects
- Chapter 4. Branches
REFERENCES
Image Sources I

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

http://www.zybooks.com/