COMP 110
TYPES

Instructor: Prasun Dewan
Prerequisites

- Interfaces
Objects vs. Primitive Types

- Instances of classes and interfaces are objects
- All other values are primitives
- Primitive types are used to construct objects
- ~Atoms vs. molecules
OBJECT TYPES

ABMIColorCalculator

ABMIColorSpreadsheet

[ABMIColorCalculator]

[ABMIColorSpreadsheet]

Height: 1.77
Weight: 77.0
BMI: 24.577867151840145

Height: 1.77
Weight: 71.0
BMI: 22.66270867247598
**Primitive Types**

- **int**
  - $3 + 4$
  - $3 - 4$
  - ...

- **String**
  - "three" + "four"
  - "three" – "four"

Overloaded operator

int and String are different types
**Primitive Types**

Overloaded arithmetic operator

<table>
<thead>
<tr>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 / 4 = 0</td>
<td>3.0 / 4.0 = 0.75</td>
</tr>
</tbody>
</table>
Kinds of Types

Primitive types
- double
- int

Object types
- String
- ABMISpreadsheet
- ABMICalculator
- AnotherBMISpreadsheet
- BMISpreadsheet

Lower case (by convention)
Upper case (by convention)
Abstract Value vs. Syntax

Representing the abstract value 2.2

2.2
0.22+E1
LBS_IN_KGS
02.2
SYNTAX FOR INVOKING ABSTRACT OPERATION

- **binary, infix**: profit - earnings
- **uniary, prefix**: - earnings
- **arbitrary, method invocation**: Math.round(bmi)
Type rules define which of these are legal.

```cpp
double height = 1.77;

int weight = 70;

double height = 2;

int weight = 70.0;

double weight = "seventy";
```
PRIMITIVE TYPES

- Each primitive type defines:
  - Range of abstract values of the type
  - Constants (literals & named constants) denoting their values
  - Operations (with invocation syntax) that can be invoked on the values of that type
  - What types can be assigned to variables of the type
There are only 10 types of people in this world:
those who read binary and those who don’t.
DOUBLE RANGE & CONSTANTS

Mathematical Real Numbers

double

Double.MIN_VALUE

Double.MAX_VALUE

2.2
0.22
2

02.20
2.
.2

0.22E+1
0.22E0

.22E1
.22E-1

exXy = x*10y
mantissa
exponent
standard

64 bits
Other Integer Subsets

Mathematical Integers \([-\infty \ldots +\infty]\)

- **byte**
  - \([-2^7 \ldots 2^7 - 1]\)
  - 8 bits
  - Byte.MIN_VALUE
  - Byte.MAX_VALUE

- **short**
  - \([-2^{15} \ldots 2^{15} - 1]\)
  - 16 bits
  - Short.MIN_VALUE
  - Short.MAX_VALUE

- **long**
  - \([-2^{63} \ldots 2^{63} - 1]\)
  - 64 bits
  - Long.MIN_VALUE
  - Long.MAX_VALUE
FLOAT SIZE & CONSTANTS

Mathematical Real Numbers

float \{-2^{31} \ldots +2^{31} - 1\}

Float.MIN_VALUE

Float.MAX_VALUE

32 bits
Mixed Assignment

long l = 70;

double d = 70;

Safe and automatically converted

int & long

int & double

double d = 70.0;
Cast

```java
int i = (int) 70.6;

int i = 70;

float f = (float) 70.6;
```

Not automatically converted

float ⊆ double

int ⊆ double
public class AnIntBMISpreadsheet implements IntBMISpreadsheet {
   int height, weight; public AnIntBMISpreadsheet() { }
   public AnIntBMISpreadsheet(
      int theInitialHeight, int theInitialWeight) {
      setHeight(theInitialHeight);
      setWeight(theInitialWeight);
   }
   public int getWeight() {  
      return weight;
   }
   public void setWeight(int newWeight) {  
      weight = newWeight;
   }
   public int getHeight() {  
      return height;
   }
   public void setHeight(int newHeight) {  
      height = newHeight;
   }
   public int getBMI() {  
      return weight/(height*height);
   }
}
CAST IN BMI SPREADSHEET

ObjectEditor.edit(new AnIntBMISpreadsheet ((int) 1.77, 75));
ASSIGNMENT RULES

Narrower than

\[ T^E \sqsupseteq T^V \]

\[ v = e \]

Wider than

\[ T^E \sqsubseteq T^V \]

\[ v = (T^V) e \]

\[ \text{double } d = 5; \]

\[ \text{bool } b = (\text{bool}) 5; \]

\[ ! (T^E \sqsupseteq T^V \mid \mid T^E \sqsubseteq T^V) \]

\[ \text{int } i = (\text{int}) 5.7; \]
ASSIGNMENT RULES FOR PRIMITIVE TYPES

- If T1 narrower than T2 (Set of instances of T1 ⊆ Set of instances of T2)
- Expression of type T1 can be assigned to Variable of type T2
- Expression of type T2 can be assigned to Variable of type T1 with cast
**Actual Parameter Assignment**

```java
double weight;

public void setWeight(double newWeight) {
    weight = newWeight;
}
```

```java
double newWeight = 70.0;
setWeight(70);
```

*Implicit assignment*
**Actual Parameter Assignment**

```java
int weight;

public void setWeight(int newWeight) {
    weight = newWeight;
}

setWeight(70.6);

int newWeight = 70.6;
```

Implicit assignment
int weight;

public void setWeight(int newWeight) {
    weight = newWeight;
}

setWeight((int)70.6);
RETURNING A VALUE

double weight;

public double getIntWeight() {
    return weight;
}
Translated into Assignment

double weight;

public int getIntWeight() {
    int getIntWeight = weight;
    return getIntWeight;
}
double weight;

public int getIntWeight() {
    return (int) weight;
}
PRIMITIVE TYPES

- Constants (Literals & Named Constants)
- Assignment Rules
- Operations with Invocation Syntax
## INT ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Operand &amp; Result Type (Signature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>add</td>
<td>int, int → int</td>
</tr>
<tr>
<td>-</td>
<td>subtract</td>
<td>int, int → int</td>
</tr>
<tr>
<td>-</td>
<td>negate</td>
<td>int → int</td>
</tr>
<tr>
<td>*</td>
<td>multiply</td>
<td>int, int → int</td>
</tr>
<tr>
<td>/</td>
<td>int quotient</td>
<td>int, int → int</td>
</tr>
<tr>
<td>%</td>
<td>int remainder</td>
<td>int, int → int</td>
</tr>
</tbody>
</table>

\[5/2 \rightarrow 2\]
\[5\%2 \rightarrow 1\]

\[x == (x/y)\times y\] (Wrong)
\[x == (x/y)\times y + (x\%y)\] (Correct)
## DOUBLE ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Operand &amp; Result Type (Signature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>add</td>
<td>double, double → double</td>
</tr>
<tr>
<td>-</td>
<td>subtract</td>
<td>double, double → double</td>
</tr>
<tr>
<td>-</td>
<td>negate</td>
<td>double → double</td>
</tr>
<tr>
<td>*</td>
<td>multiply</td>
<td>double, double → double</td>
</tr>
<tr>
<td>/</td>
<td>int quotient</td>
<td>double, double → double</td>
</tr>
</tbody>
</table>

\[
5.0/2.0 \rightarrow 2.5
\]
OVERFLOW

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer.MAX_VALUE + 1</td>
<td>Integer.MAX_VALUE</td>
</tr>
<tr>
<td>Integer.MIN_VALUE - 1</td>
<td>Integer.MIN_VALUE</td>
</tr>
<tr>
<td>(double) Integer.MIN_VALUE - 1.0</td>
<td>(double) (Integer.MIN_VALUE - 1.0)</td>
</tr>
<tr>
<td>Double.MAX_VALUE + 1</td>
<td>Double.MAX_VALUE</td>
</tr>
<tr>
<td>Double.MIN_VALUE - 1</td>
<td>Double.MIN_VALUE</td>
</tr>
</tbody>
</table>
Divide by Zero

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/0</td>
<td>Exception</td>
</tr>
<tr>
<td>-10/0</td>
<td>Exception</td>
</tr>
<tr>
<td>10.0/0</td>
<td>Double.POSITIVE_INFINITY</td>
</tr>
<tr>
<td>-10.0/0</td>
<td>Double.NEGATIVE_INFINITY</td>
</tr>
<tr>
<td>0/0</td>
<td>Exception</td>
</tr>
<tr>
<td>0.0/0.0</td>
<td>Double.NaN</td>
</tr>
</tbody>
</table>
INT DIVIDE BY ZERO

class java.lang.ArithmeticException
java.lang.ArithmeticException: / by zero
    at lectures.types.AnIntBMISpreadsheet.getBMI(AnIntBMISpreadsheet.java:27)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
DOUBLE OVERFLOW
## Mixed Operations

<table>
<thead>
<tr>
<th>Narrower type converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/2.0</td>
</tr>
<tr>
<td>→</td>
</tr>
<tr>
<td>5.0/2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int i = (int) (5/2.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i = (int) (5.0/2.0)</td>
</tr>
<tr>
<td>int i = (int) (2.5)</td>
</tr>
<tr>
<td>int i = 2</td>
</tr>
</tbody>
</table>

- **double d = 5/(int)2.0**
- **double d = 5/2**
- **double d = 2**
- **double d = 2.0**
**Strong vs. Weak Typing**

- **"hello" - 1**
  - Legal under weak typing
  - "anything goes"

- **int minus**
  - Illegal under strong typing
  - "strict type rules"
# Miscellaneous Math Operations

<table>
<thead>
<tr>
<th>Operations (invoked on Math)</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs()</td>
<td>double → double, int → int</td>
</tr>
<tr>
<td>acos(), asin(), atan()</td>
<td>double → double</td>
</tr>
<tr>
<td>cos(), sin(), tan()</td>
<td></td>
</tr>
<tr>
<td>pow()</td>
<td>double, double → double</td>
</tr>
<tr>
<td>exp(), log()</td>
<td>double → double</td>
</tr>
<tr>
<td>round()</td>
<td>double → long</td>
</tr>
<tr>
<td>random(), pi()</td>
<td>→ double</td>
</tr>
<tr>
<td>sqrt()</td>
<td>double → double</td>
</tr>
</tbody>
</table>

- Math.PI → π
- Math.pow(5, 3) → 5³
- Math.round(5.9) → 6
- (int) 5.9 → 5
- int i = (int) Math.Round(5.9) → 6
BOOLEAN CONSTANTS

```
boolean
true
false
```
## Relational Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Signature of int Implementation</th>
<th>Signature of double Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal?</td>
<td>int, int → boolean</td>
<td>double, double → boolean</td>
</tr>
</tbody>
</table>

### Examples

- `5 == 5` → true
- `5 != 5` → false
- `5 == 4` → false
- `5 != 4` → true
- `5 >= 4` → true
- `5 <= 4` → false
# BOOLEAN OPERATIONS

<table>
<thead>
<tr>
<th>Name(s)</th>
<th>Action</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>boolean → boolean</td>
</tr>
<tr>
<td>&amp;&amp;, &amp;</td>
<td>and</td>
<td>boolean, boolean → boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
</tbody>
</table>

- !true → false
- !false → true
- true && true → true
- true && false → false
- false && true → false
- false && false → false
**Short-Circuit Evaluation**

<table>
<thead>
<tr>
<th>Name(s)</th>
<th>Action</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>boolean → boolean</td>
</tr>
<tr>
<td>&amp;&amp;, &amp;</td>
<td>and</td>
<td>boolean, boolean → boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
</tbody>
</table>

Short-circuit evaluation

- false && (9654.34/323.13 > 32.34) → false
- true || (9654.34/323.13 > 32.34) → false

Second operand not evaluated

Regular evaluation

- false & (9654.34/323.13 > 32.34) → false
- true | (9654.34/323.13 > 32.34) → false

Second operand evaluated
# Short-Circuit Evaluation

<table>
<thead>
<tr>
<th>Name(s)</th>
<th>Action</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>boolean $\rightarrow$ boolean</td>
</tr>
<tr>
<td>&amp;&amp;, &amp;</td>
<td>and</td>
<td>boolean, boolean $\rightarrow$ boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
</tbody>
</table>

$$false \&\& (10/0 == Integer.MAX\_VALUE) \rightarrow false$$

$$false \& (10/0 == Integer.MAX\_VALUE)$$

An error in some programming languages
Complex Expressions

false && (10/0 == Integer.MAX_VALUE)

false && (10/0 == Integer.MAX_VALUE)

Operator evaluation order?
**Complex Expressions**

false && (10 / 0)

Sub-expression

false && (10 / 0)

Operator evaluation order?
**BOOLEAN VS. NUMBER EXPRESSIONS**

```java
boolean overWorked = hoursWorked > MAX_HOURS
```

True if `hoursWorked` is greater than `MAX_HOURS` and false otherwise.

```java
int earnings = hourlyWage*hoursWorked + BONUS
```
Boolean Property
public boolean isOverWeight() {
}

boolean Property Code
final double HIGH_BMI = 28;

public boolean isOverWeight() {
    return getBMI() > HIGH_BMI;
}


private final double HIGH_BMI = 25;

public boolean isOverWeight() {
    return getBmi() > HIGH_BMI;
}
// declare in interface

private final double HIGH_BMI = 25;

public boolean isOverWeight() {
    return getBmi() > HIGH_BMI;
}
PREVENTING INVALID BMI
## Operator Precedence

<table>
<thead>
<tr>
<th>Unary</th>
<th>Cast</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>false &amp;&amp; 10 / 0 &gt; 0</td>
</tr>
<tr>
<td>- (T)</td>
<td>- 5 - 4</td>
</tr>
<tr>
<td>* / &amp;</td>
<td>!true &amp;&amp; false</td>
</tr>
<tr>
<td>+ -</td>
<td>5 / 4 * 3</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>true</td>
</tr>
<tr>
<td>== !=</td>
<td>false</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>(int) 5 / 2.0</td>
</tr>
<tr>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Operator Precedence (Edit)

## Unary
- `!`
- `-`
- `(T)`

## Cast
- `true` && false
- `false` || `true`
- `(int) 5 / 2.0`

## Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!</code></td>
<td>false &amp;&amp; 10 / 0 &gt; 0</td>
</tr>
<tr>
<td><code>*</code></td>
<td>- 5 - 4</td>
</tr>
<tr>
<td><code>/</code></td>
<td>!true &amp;&amp; false</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>5 / 4 * 3</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>false</td>
</tr>
<tr>
<td><code>==</code></td>
<td>(int) 5 / 2.0</td>
</tr>
<tr>
<td><code>!=</code></td>
<td></td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td></td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
</tbody>
</table>
### Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>false &amp;&amp; 10 / 0 &gt; 0</td>
<td>( false &amp;&amp; ( (10 / 0) &gt; 0 ) )</td>
</tr>
<tr>
<td>-</td>
<td>( - 5 ) - 4</td>
<td>( ( - 5 ) - 4 )</td>
</tr>
<tr>
<td>* / &amp;</td>
<td>!true &amp;&amp; false</td>
<td>( !true ) &amp;&amp; false )</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>5 / 4 * 3</td>
<td>( ( 5 / 4 ) * 3 )</td>
</tr>
<tr>
<td>== !</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>(int) 5 / 2.0</td>
<td>( ( (int) 5 ) / 2.0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Unary
- !
- -
- (T)

#### Cast
- *
- / &
- + -
- < > <= >=
- == !=
- &&
- |
- &
- &&
- ||

#### Order of Operations

1. Unary
2. Cast
3. !
4. -
5. * / &
6. + -
7. < > <= >=
8. == !=
9. &&
10. |
11. ||
Printing Arbitrary Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.out.println (2)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>System.out.println (2.0)</code></td>
<td>2.0</td>
</tr>
<tr>
<td><code>System.out.println ((int) 2.0)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>System.out.println (5 &gt; 0)</code></td>
<td>true</td>
</tr>
</tbody>
</table>