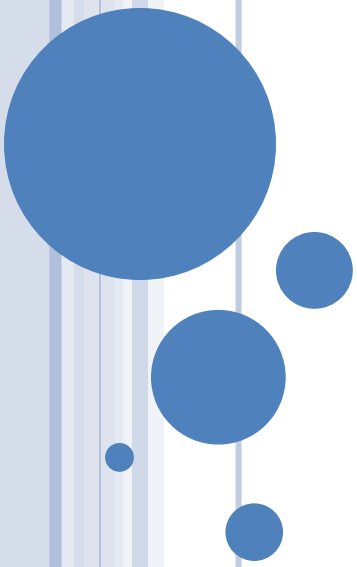


COMP 110
TYPES

Instructor: Prasun Dewan

Prerequisites

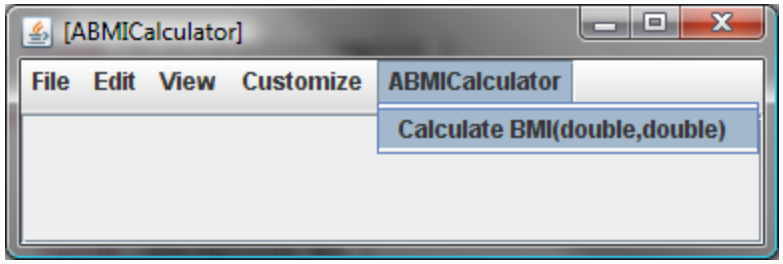
- Interfaces

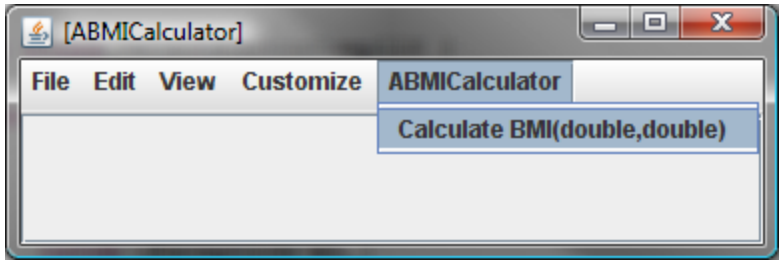
Objects vs. Primitive Types

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

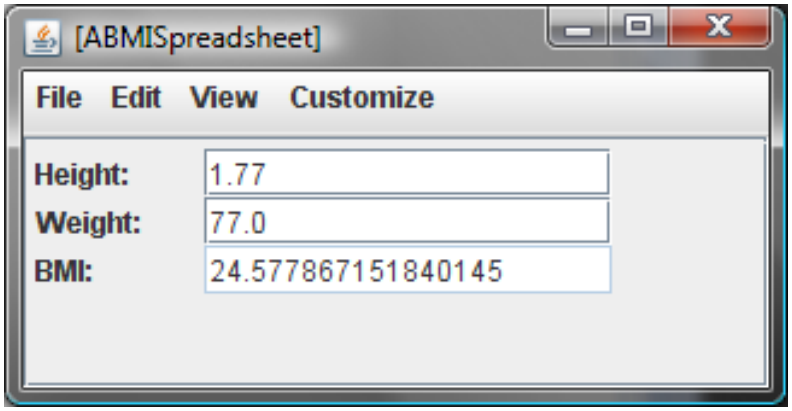
Object Types

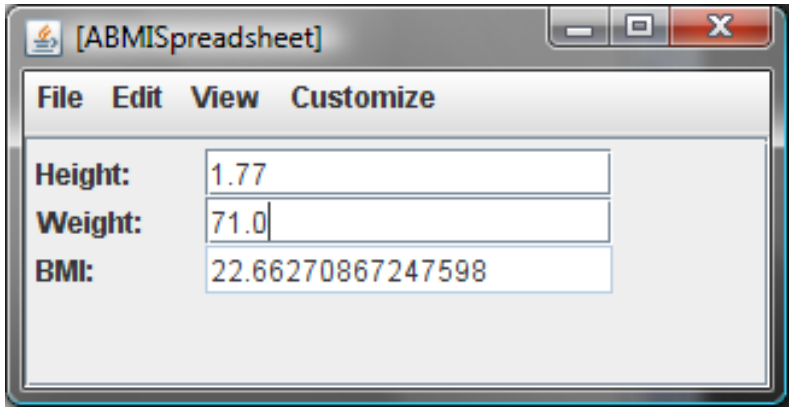
ABMICalculator



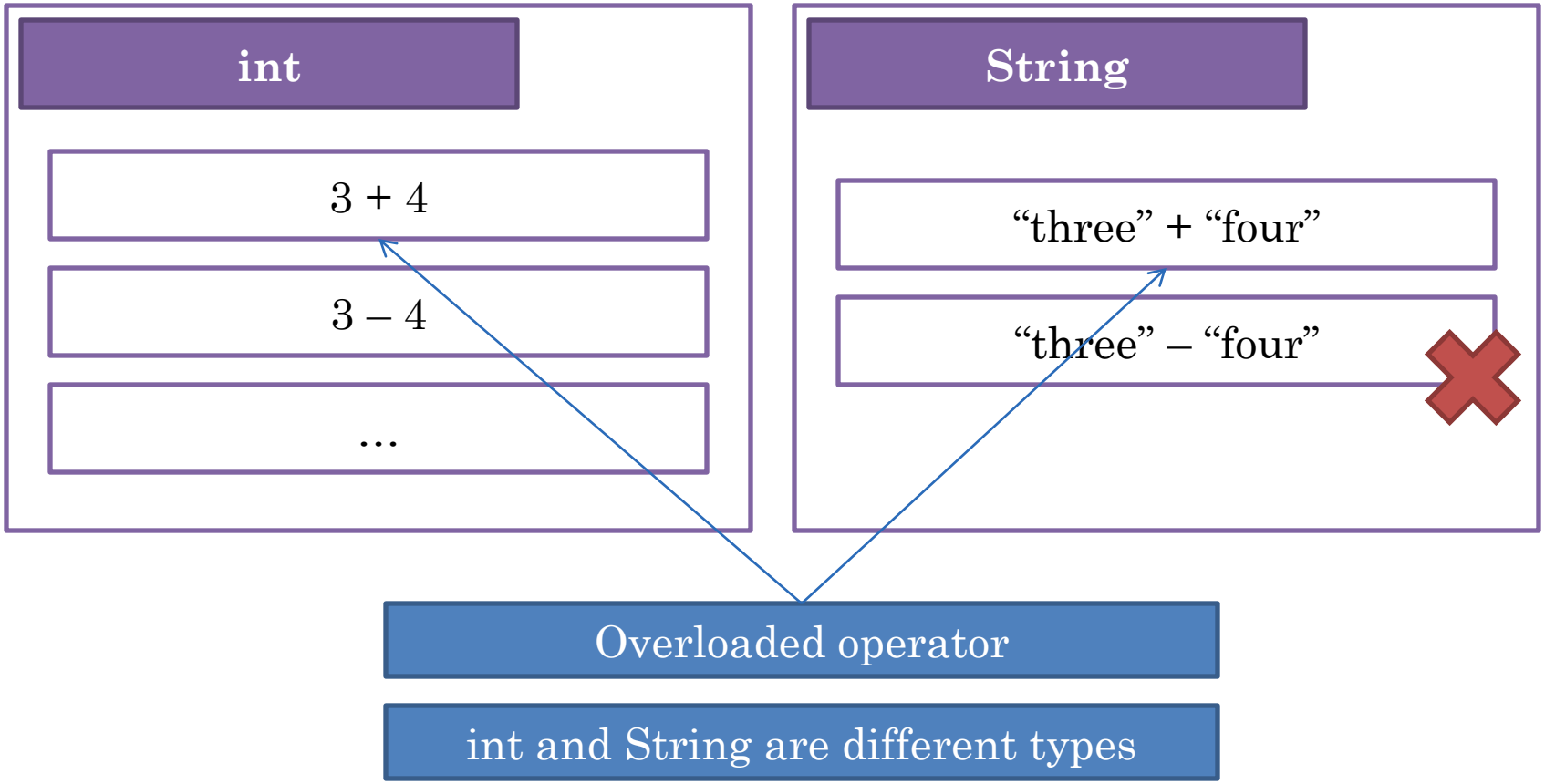


ABMISpreadsheet

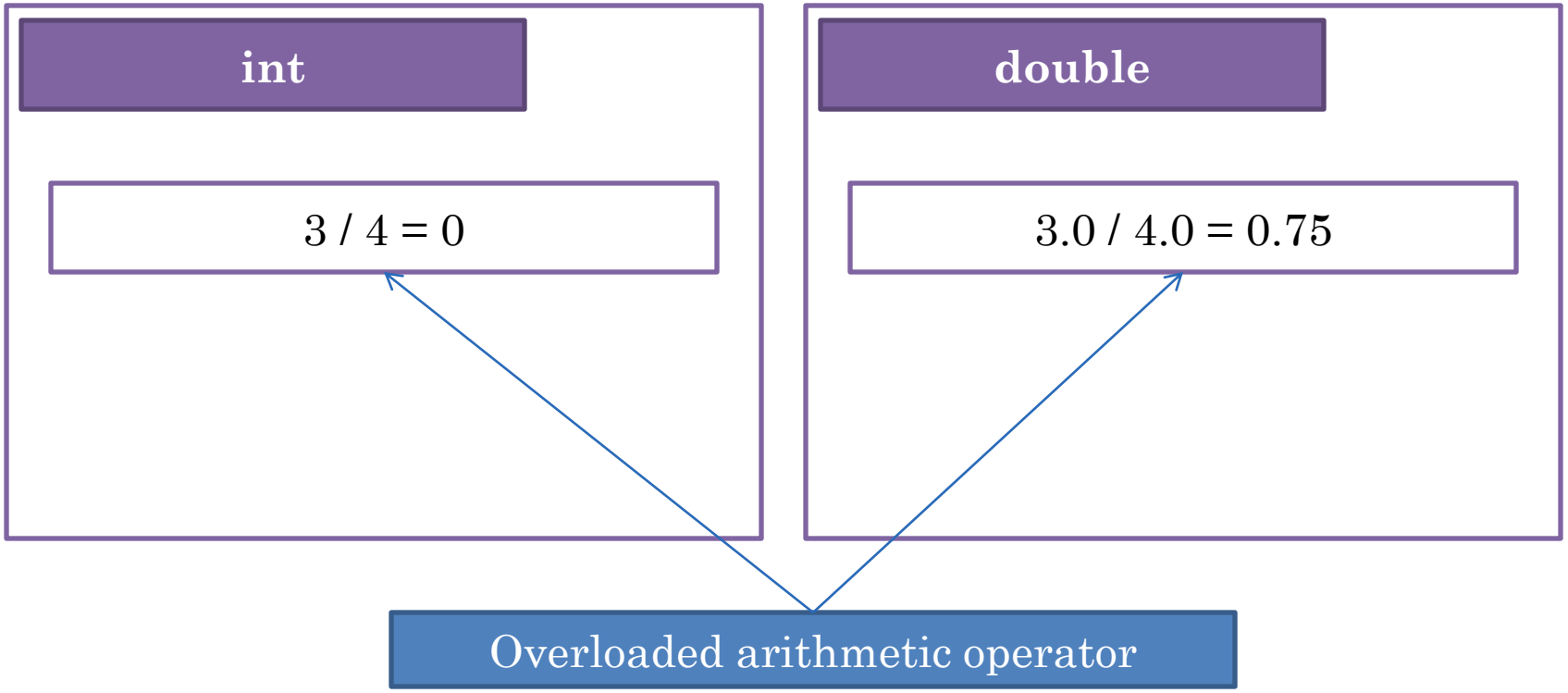




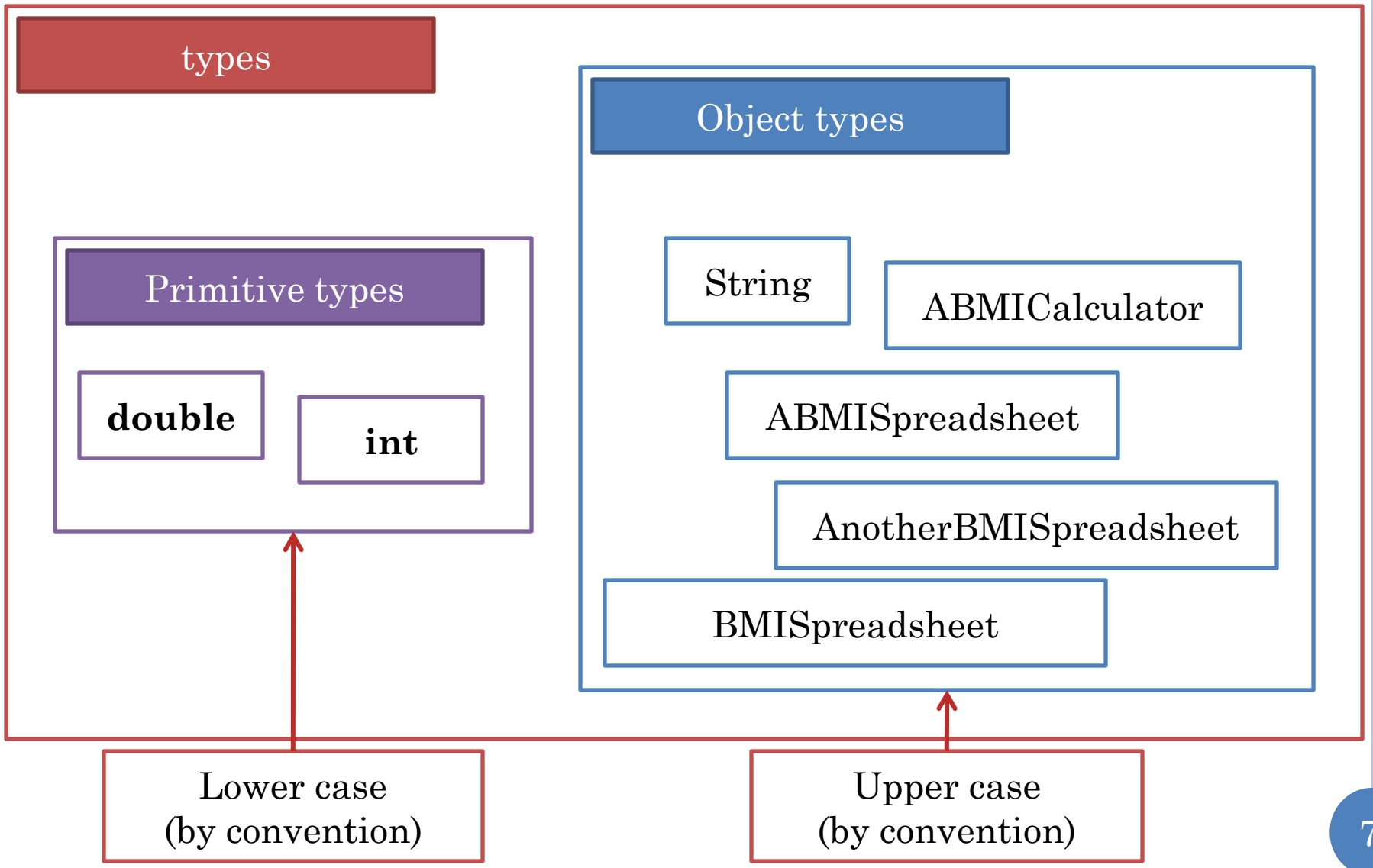
Primitive Types



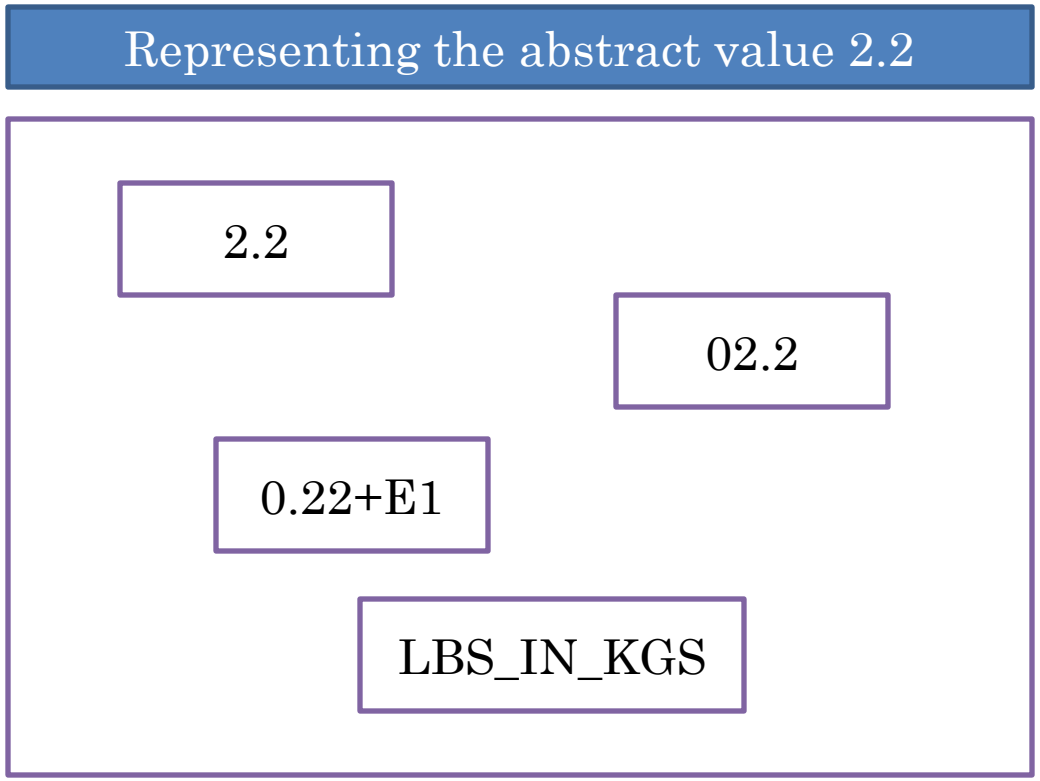
Primitive Types



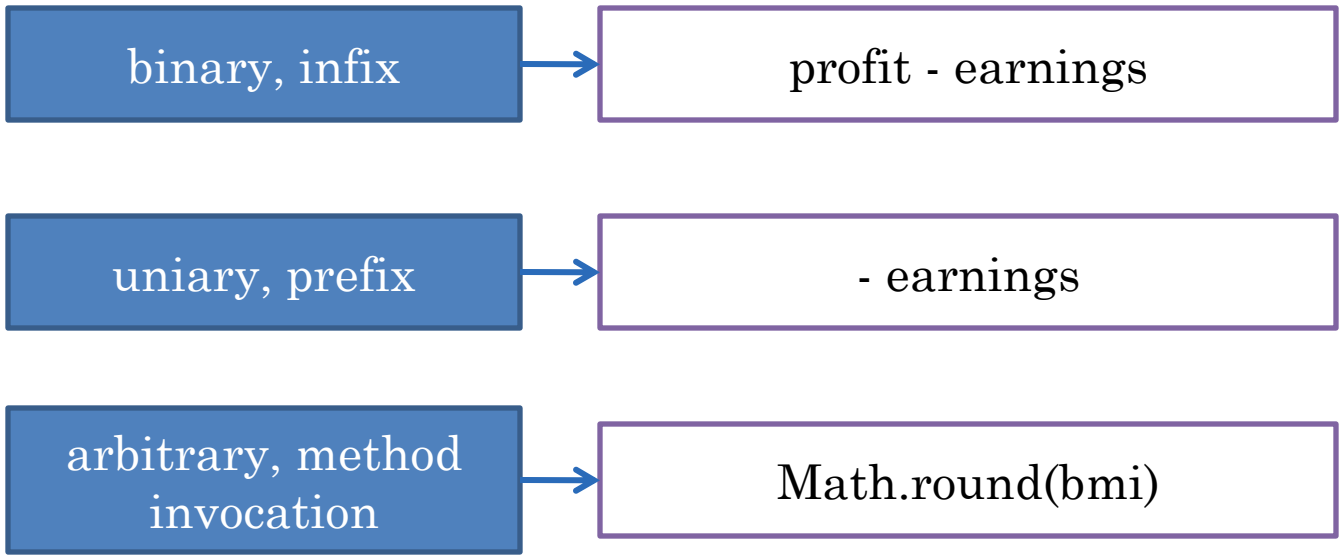
Kinds of Types



Abstract Value vs. Syntax



Syntax for Invoking Abstract Operation



Type Rules Regarding Assignment

```
double height = 1.77;
```

```
int weight = 70;
```

```
double height = 2;
```

```
int weight = 70.0;
```

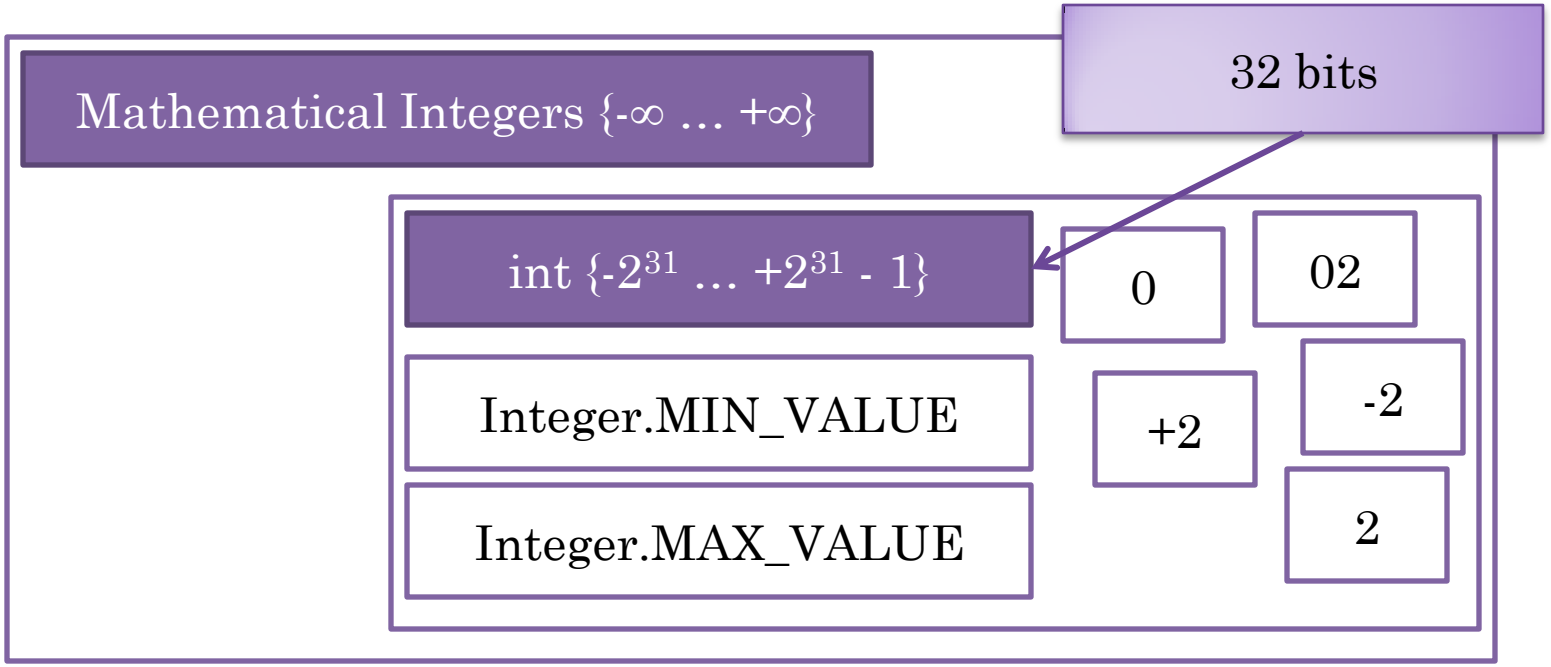
```
double weight = "seventy";
```

Type rules define which of these are legal.

Primitive Types

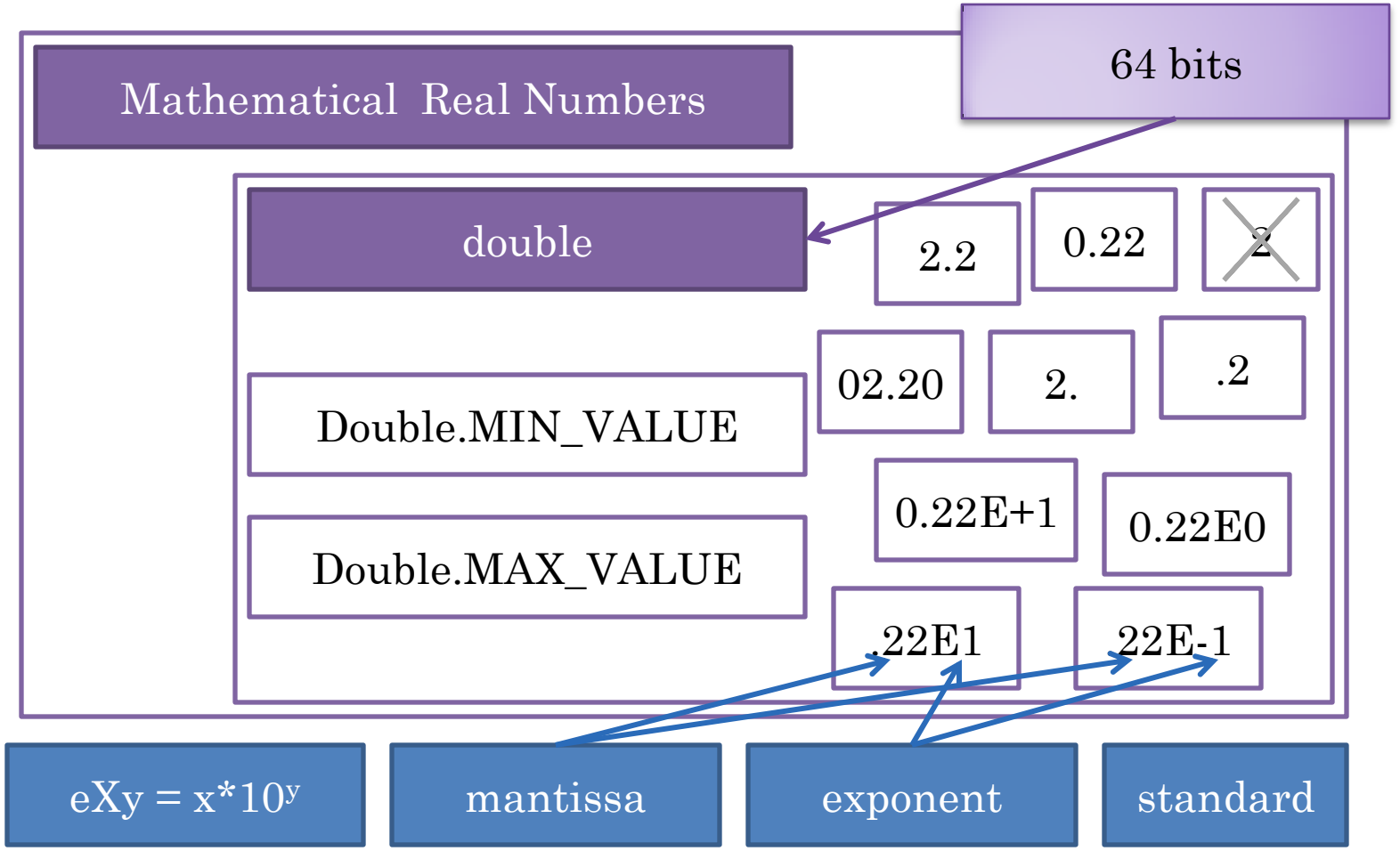
- Each primitive type defines:
  - Range of abstract values of the type
  - Constants (literals & named constants) denoting their values
  - Operations (with invocation syntax) that can be invoked on the values of that type
  - What types can be assigned to variables of the type

int Range & Constants

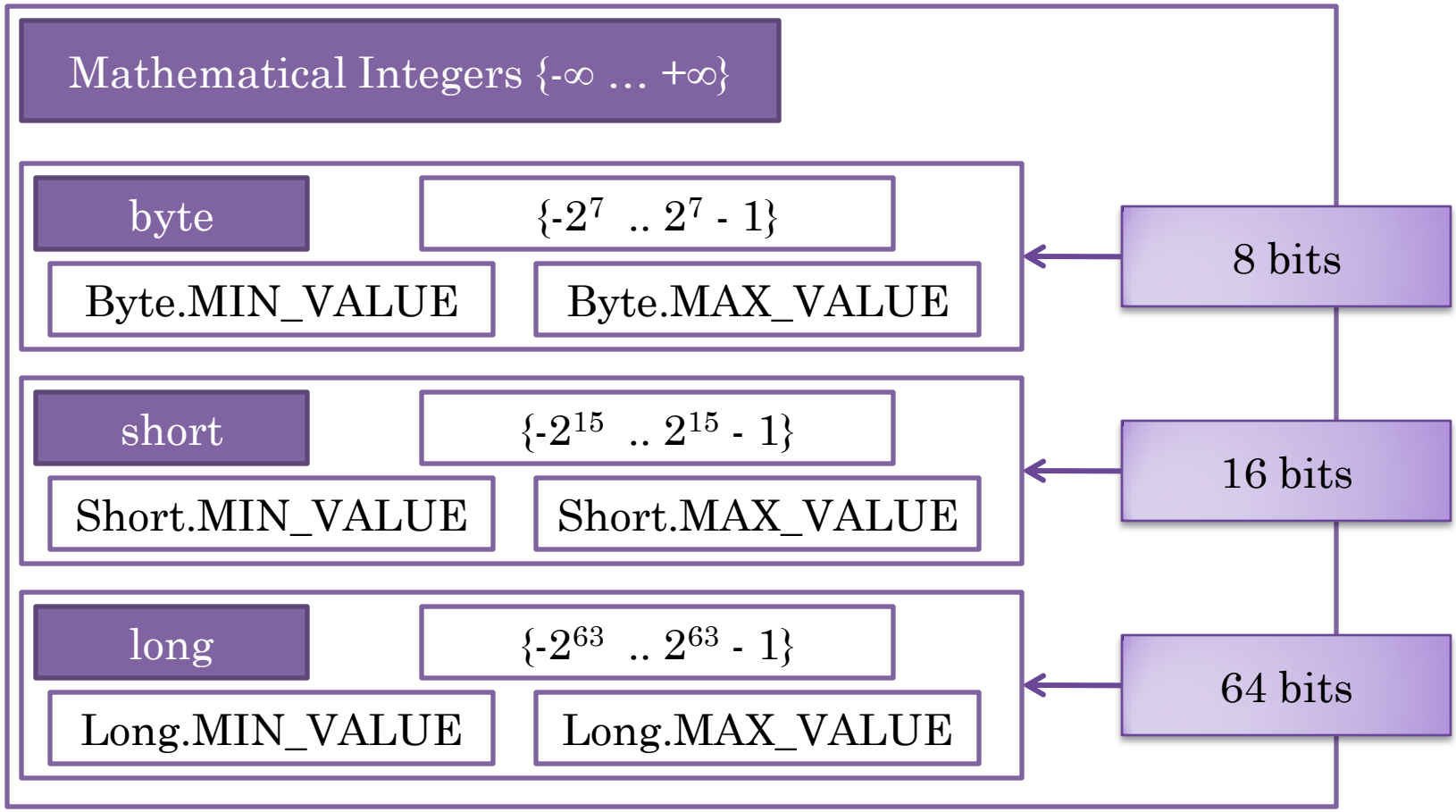


There are only 10 types of people in this world:
those who read binary and those who don't.

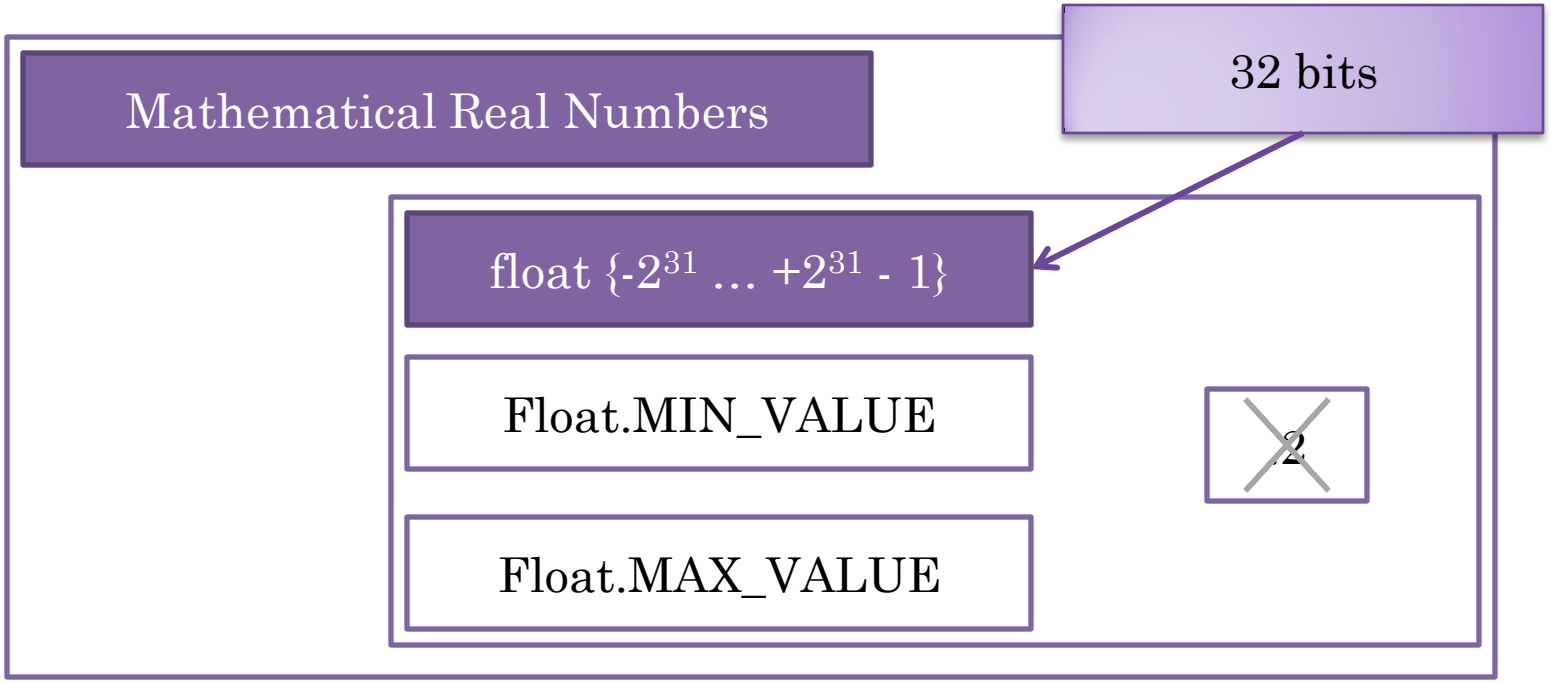
DOUBLE RANGE & CONSTANTS



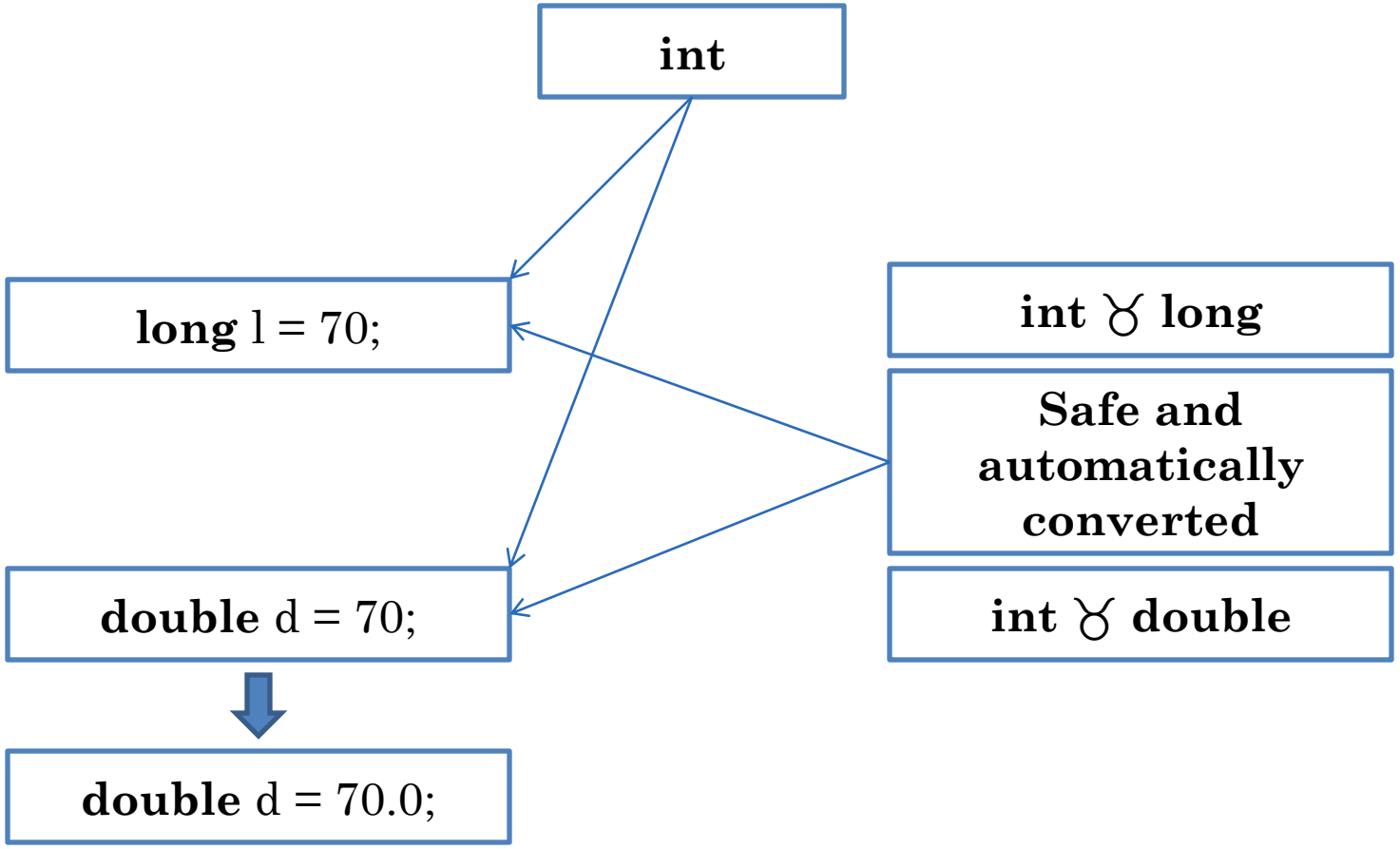
Other Integer Subsets



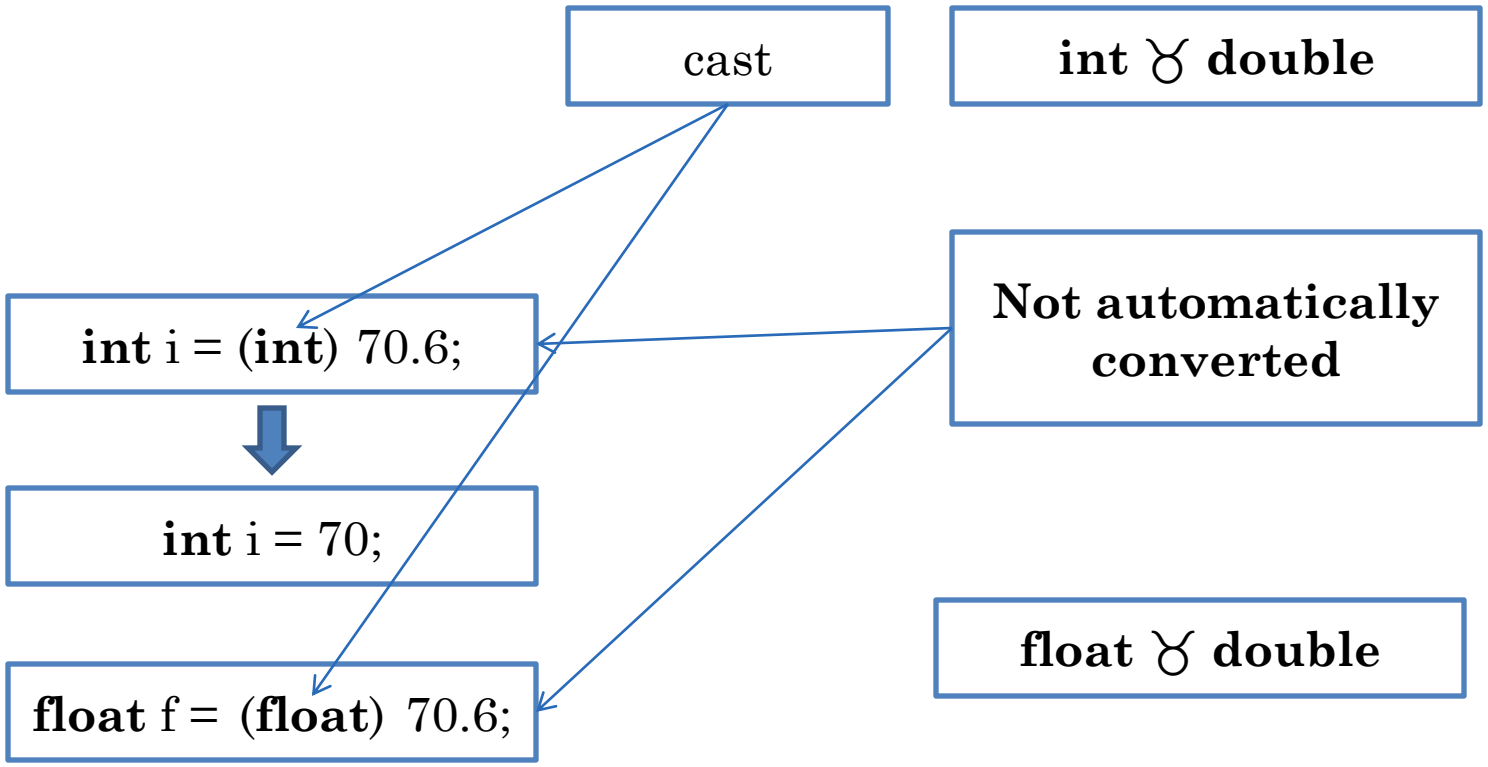
FLOAT SIZE & CONSTANTS



Mixed Assignment



Cast

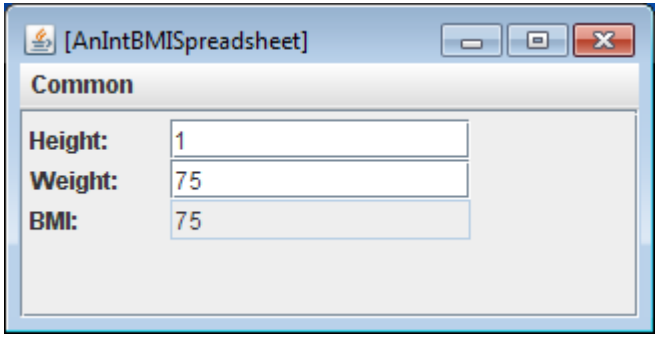

cast
int ♉ double
int i = (int) 70.6;
Not automatically converted
int i = 70;
float ♉ double
float f = (float) 70.6;

INT BMI SPREADSHEET

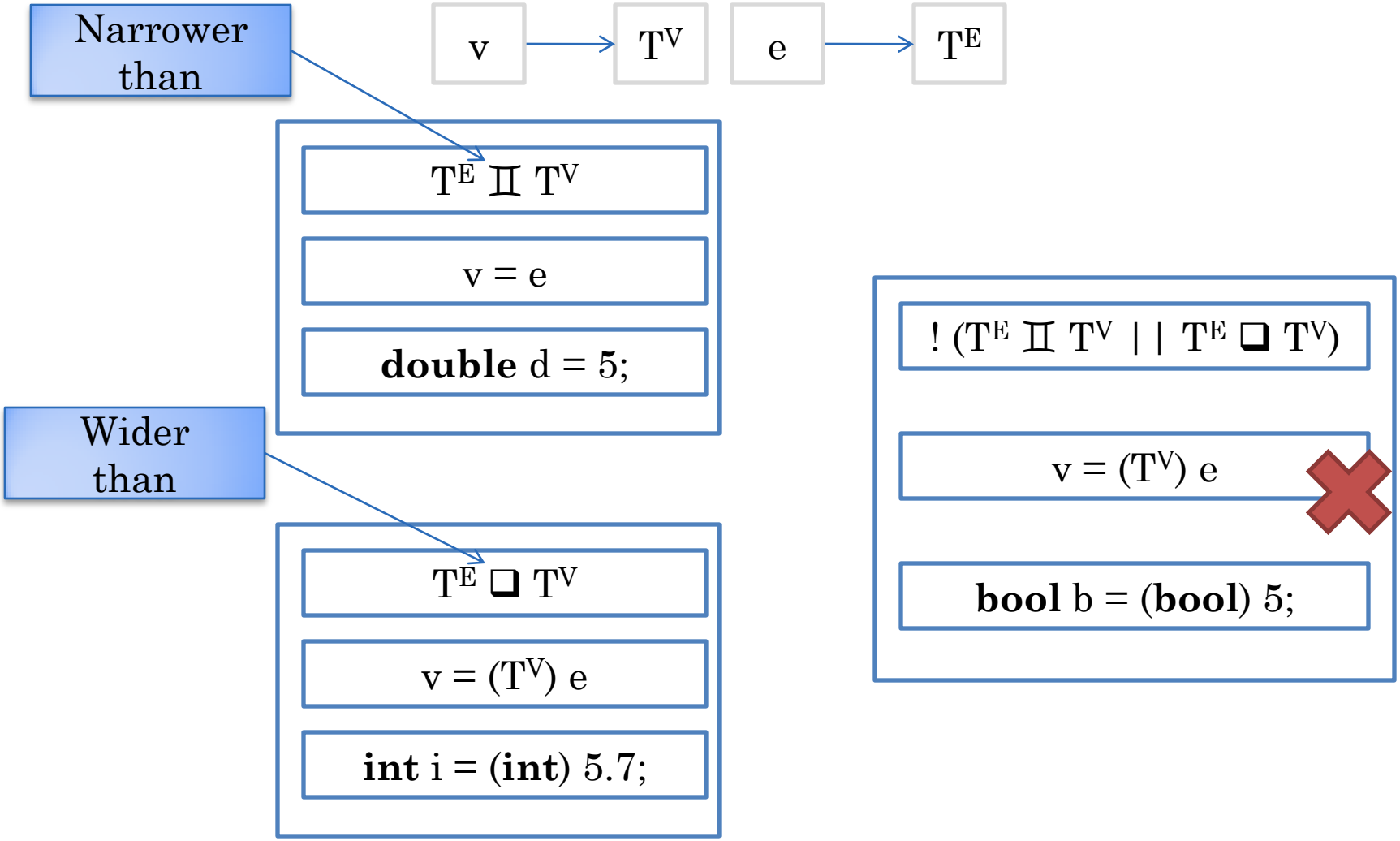
```
public class AnIntBMISpreadsheet implements IntBMISpreadsheet {
  int height, weight; public AnIntBMISpreadsheet() { }
    public AnIntBMISpreadsheet(
    int theInitialHeight, int theInitialWeight) {
    setHeight(theInitialHeight);
    setWeight(theInitialWeight);
  }
  public int getWeight() {
    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



Assignment Rules for Primitive Types

- If T1 narrower than T2 (Set of instances of T1 $\subseteq$ 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

```
double weight;

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

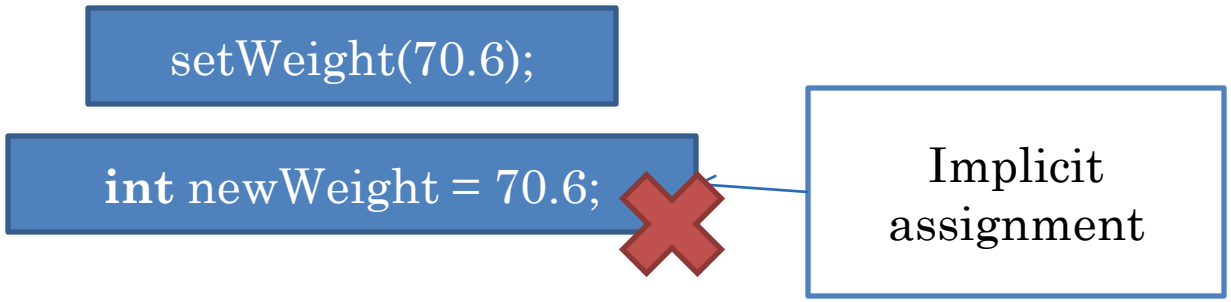
setWeight(70);

double newWeight = 70.0;

Implicit assignment

Actual Parameter Assignment

```
int weight;

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



Actual Parameter Assignment

```
int weight;

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

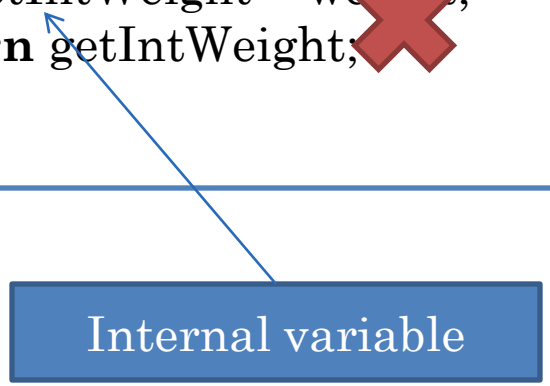
```
setWeight((int)70.6);
```

Returning a Value

```
double weight;

public double getIntWeight() {
        return weight;
}
```

Translated into Assignment

```
double weight;

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



Translated into Assignment

```
double weight;

public int getIntWeight() {
        return (int) weight;
}
```

Primitive Types

- Constants (Literals & Named Constants)
- Assignment Rules
- Operations with Invocation Syntax

INT ARITHMETIC OPERATIONS

Name	Action	Operand & Result Type (Signature)
+	add	int, int → int
-	subtract	int, int → int
-	negate	int → int
*	multiply	int, int → int
/	int quotient	int, int → int
%	int remainder	int, int → int

5/2 → 2

5%2 → 1

x == (x/y)*y 

x == (x/y)*y + (x%y)

DOUBLE ARITHMETIC OPERATIONS

Name	Action	Operand & Result Type (Signature)
+	add	double, double→ double
-	subtract	double, double→ double
-	negate	double→ double
*	multiply	double, double→ double
/	int quotient	double, double→ double

5.0/2.0 → 2.5

Overflow

Integer.MAX_VALUE + 1	→	Integer.MAX_VALUE
Integer.MIN_VALUE - 1	→	Integer.MIN_VALUE

(double) Integer.MIN_VALUE - 1.0	→	(double) (Integer.MIN_VALUE - 1.0)

Double.MAX_VALUE + 1	→	Double.MAX_VALUE
Double.MIN_VALUE - 1	→	Double.MIN_VALUE

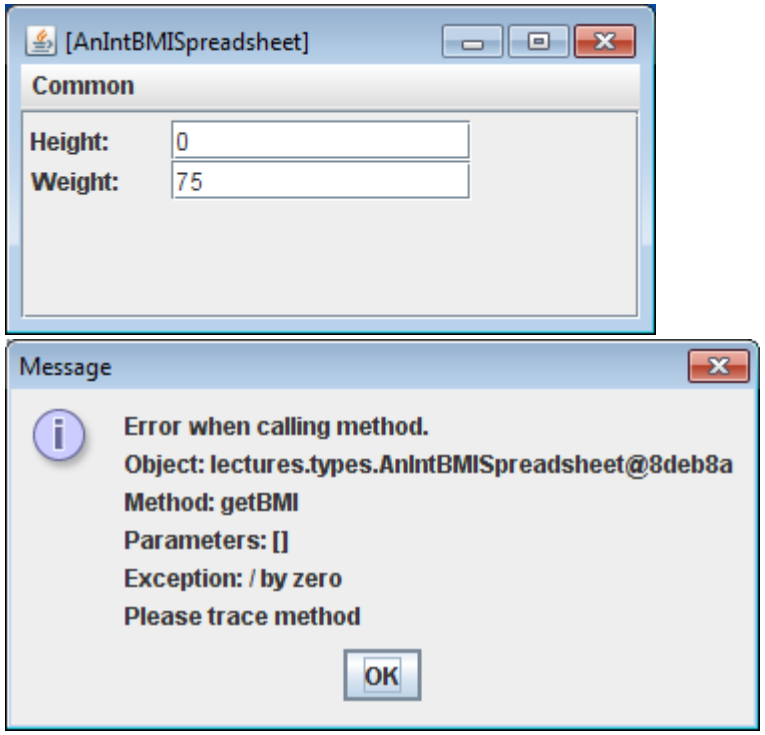
Divide by Zero

10/0	→	Exception
-10/0	→	Exception

10.0/0	→	Double.POSITIVE_INFINITY
-10.0/0	→	Double.NEGATIVE_INFINITY

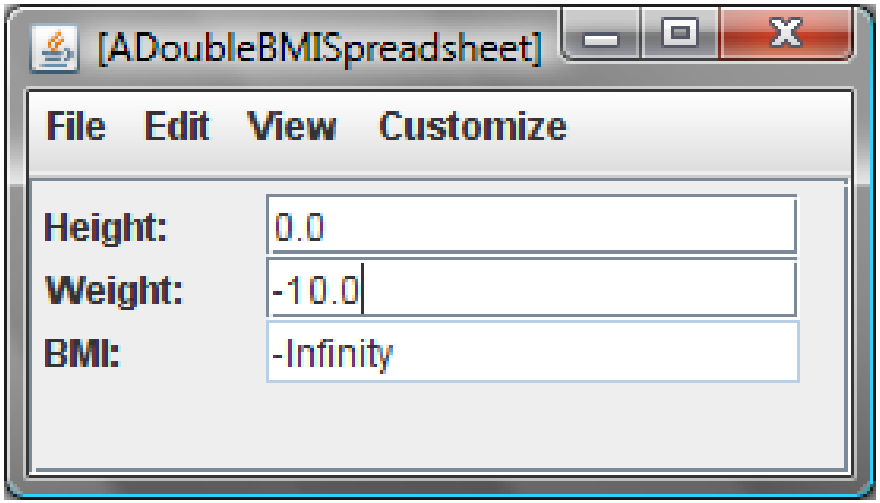
0/0	→	Exception
0.0/0.0	→	Double.NaN

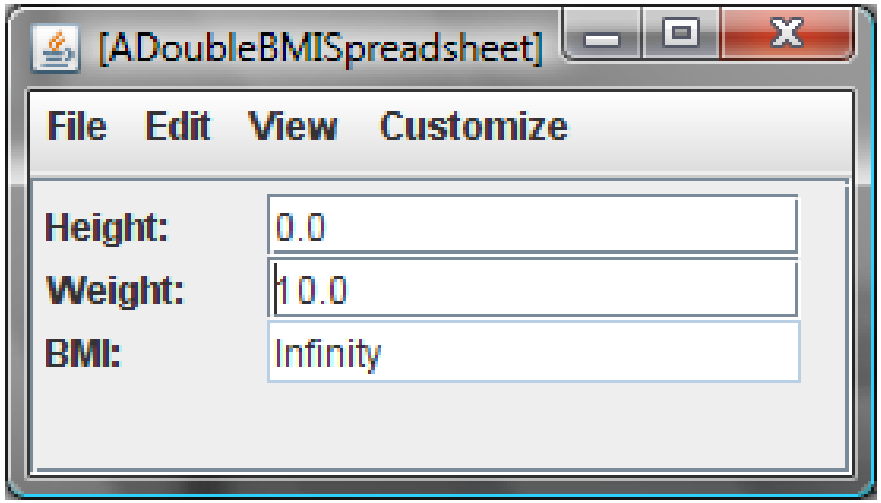
int Divide By Zero

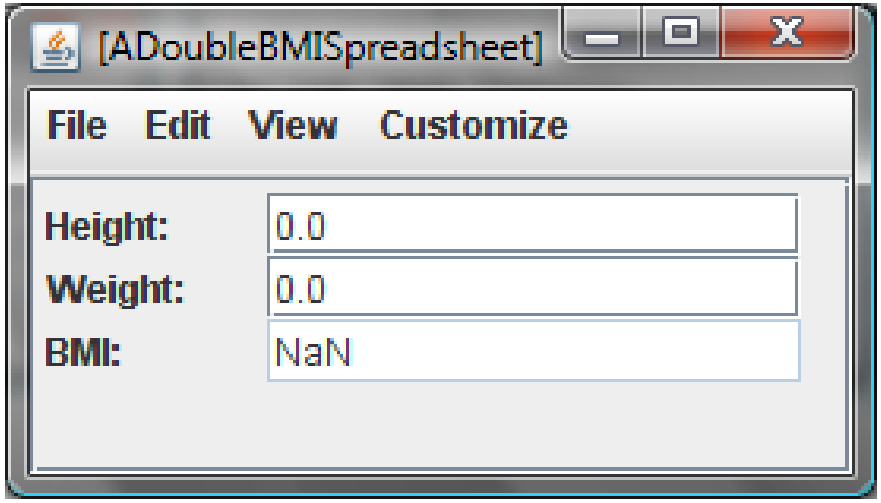



```
class java.lang.ArithmeticException
java.lang.ArithmeticException: / by zero
        at lectures.types.AnIntBMISpreadsheet.getBMI(AnIntBMISpreadsheet.java:27)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)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

DOUBLE OVERFLOW







Mixed Operations

Narrower type converted

5/2.0 → 5.0/2.0

int i = (int) (5/2.0)

int i = (int) (5.0/2.0)

int i = (int) (2.5)

int i = 2

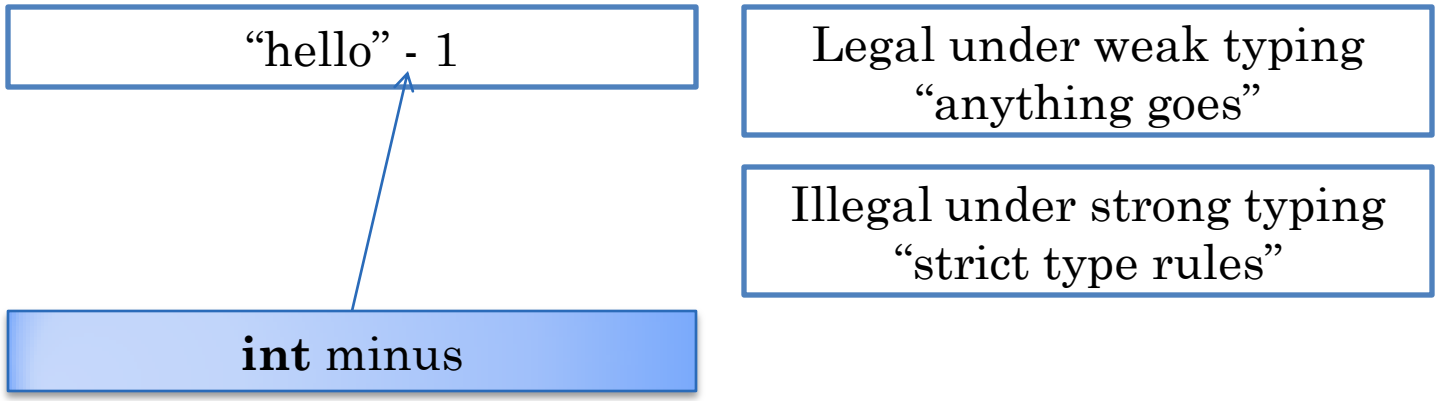
doube d = 5/(int)2.0

double d = 5/2

double d = 2

double d = 2.0

Strong vs. Weak Typing



Miscellaneous Math Operations

Operations (invoked on Math)	Signature
abs()	double → double, int → int
acos(), asin(), atan() cos(), sin(), tan()	double → double
pow()	double, double → double
exp(), log()	double → double
round()	double → long
random(), pi()	→ double
sqrt()	double → double

Math.PI → π

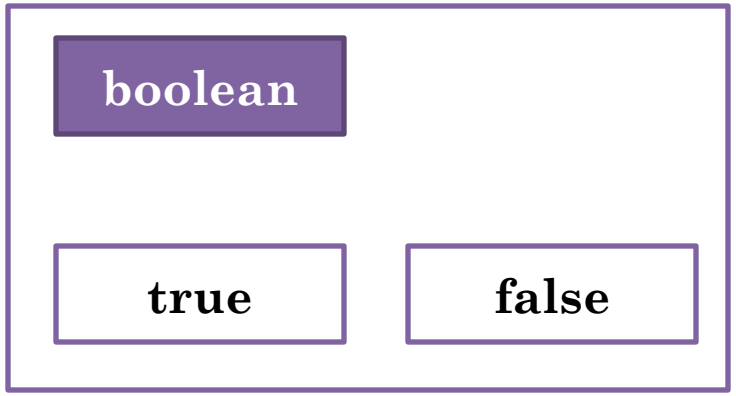
Math.pow(5, 3) → 5^3

Math.round(5.9) → 6

(int) 5.9 → 5

int i = (int) Math.Round(5.9) → 6

BOOLEAN CONSTANTS



Relational Operations

Name	Action	Signature of int Implementation	Signature of double Implementation
==	equal?	int, int → boolean	double, double → boolean
!=	not equal?	int, int → boolean	double, double → boolean
>	greater than?	int, int → boolean	double, double → boolean
<	less than?	int, int → boolean	double, double → boolean
>=	greather than or equal?	int, int → boolean	double, double → boolean
<=	less than or equal?	int, int → boolean	double, double → boolean

5 == 5 → true

5 == 4 → false

5 >= 4 → true

5 <= 4 → false

5 != 5 → false

5 != 4 → true

BOOLEAN OPERATIONS

Name(s)	Action	Signature
!	not	boolean → boolean
&&, &	and	boolean, boolean → boolean
||, |	or	boolean, boolean → boolean

!true → false

!false → true

true && true → true

true && false → false

false && true → false

false && false → false

true || true → true

true || false → true

false || true → true

false || false → false

Short-Circuit Evaluation

Name(s)	Action	Signature
!	not	boolean → boolean
&&, &	and	boolean, boolean → boolean
||, |	or	boolean, boolean → boolean

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

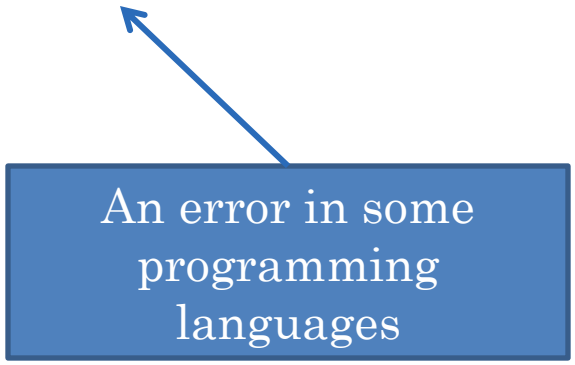
Second operand evaluated

Short-Circuit Evaluation

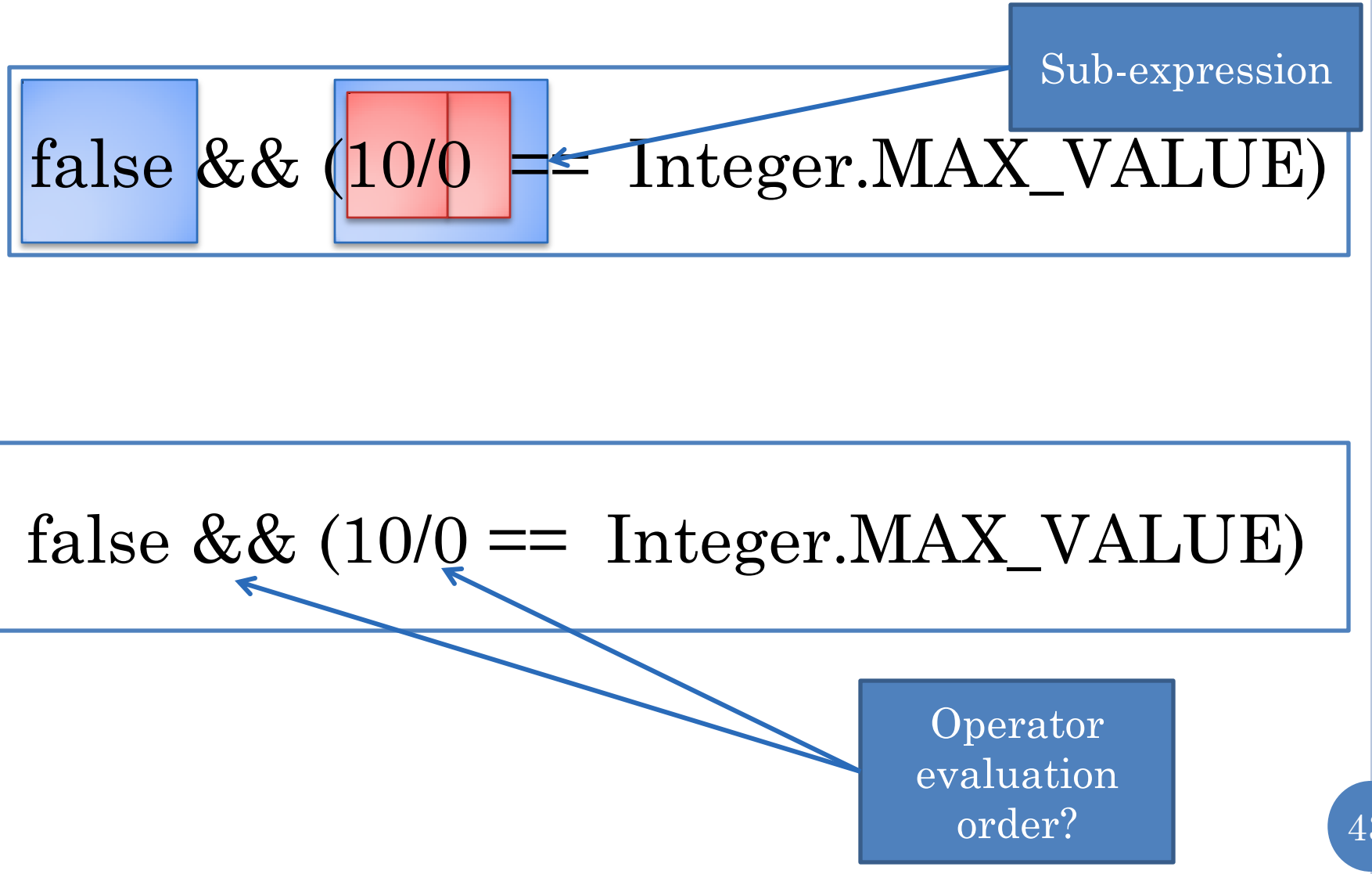
Name(s)	Action	Signature
!	not	boolean → boolean
&&, &	and	boolean, boolean → boolean
||, |	or	boolean, boolean → boolean

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

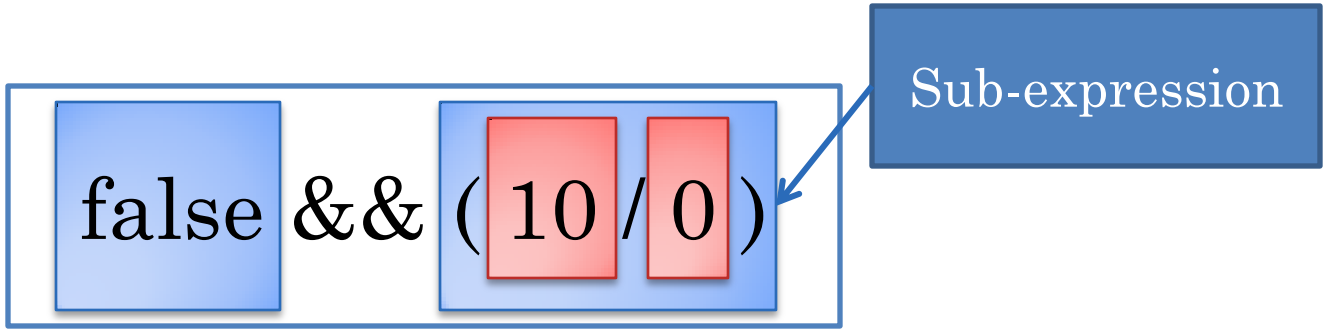
false & (10/0 == Integer.MAX_VALUE)

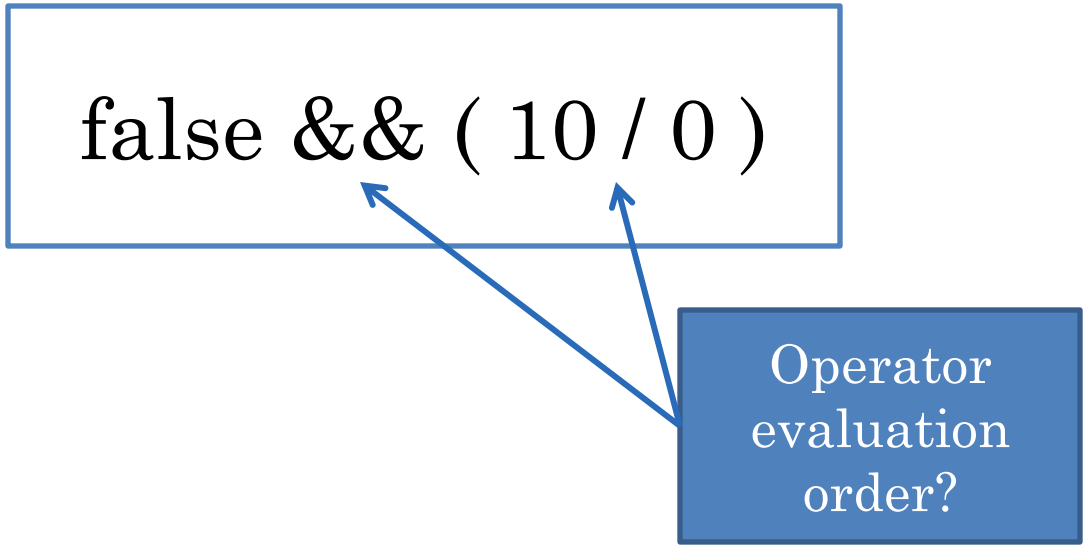


Complex Expressions



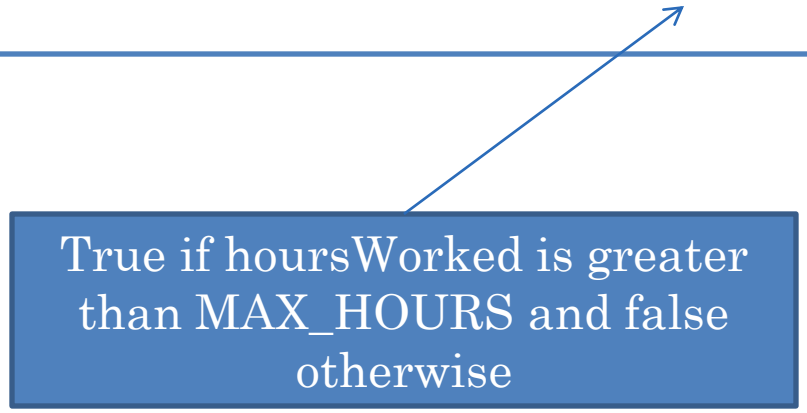
Complex Expressions





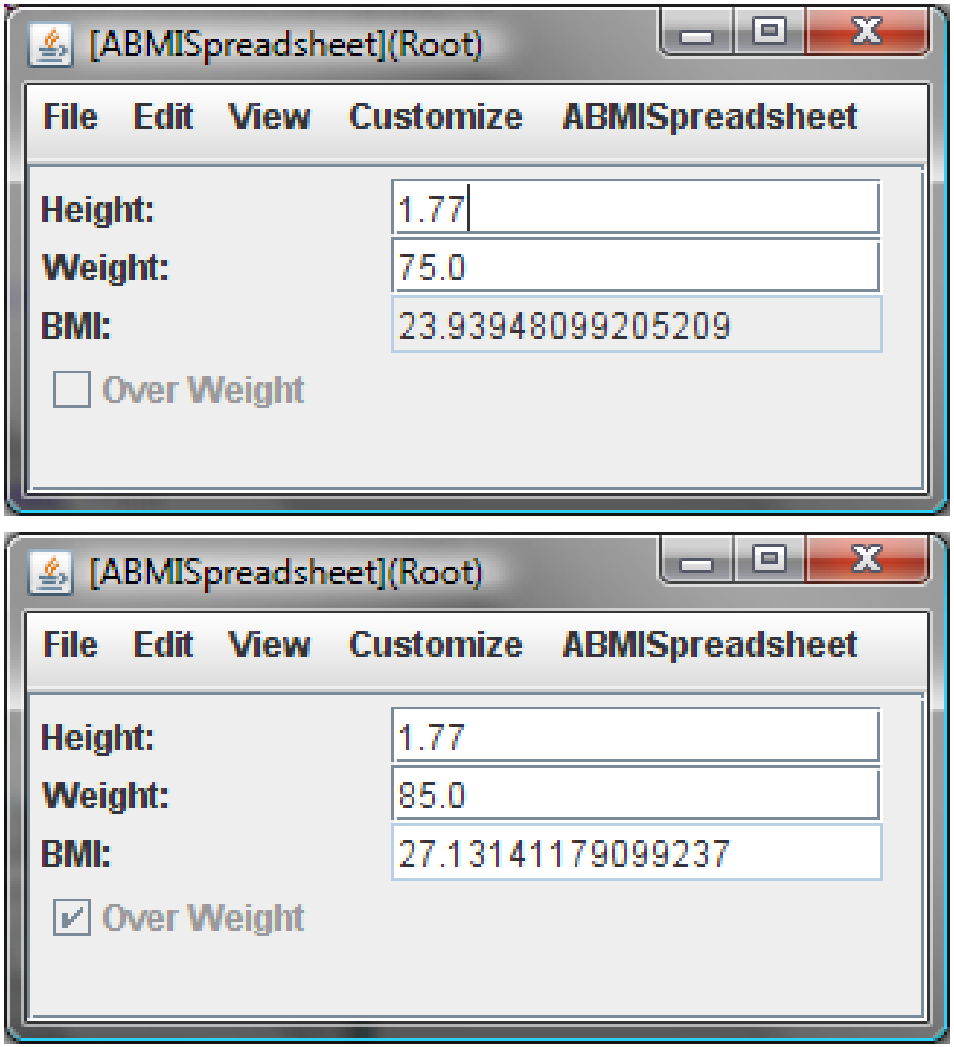
BOOLEAN VS. NUMBER EXPRESSIONS

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




```
int earnings = hourlyWage*hoursWorked + BONUS
```

BOOLEAN PROPERTY



BOOLEAN PROPERTY CODE

```
public boolean isOverWeight() {
}
```

BOOLEAN PROPERTY CODE (EDIT)

```
    final double HIGH_BMI = 28;
public boolean isOverWeight() {
        return getBMI() > HIGH_BMI;
            }
```

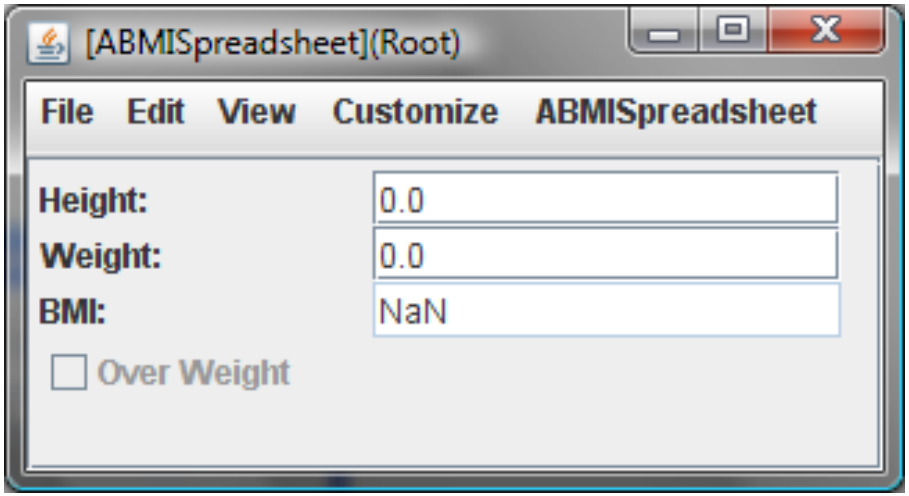
BOOLEAN PROPERTY CODE

```
private final double HIGH_BMI = 25;

public boolean isOverWeight() {
        return getBmi() > HIGH_BMI;
}
```

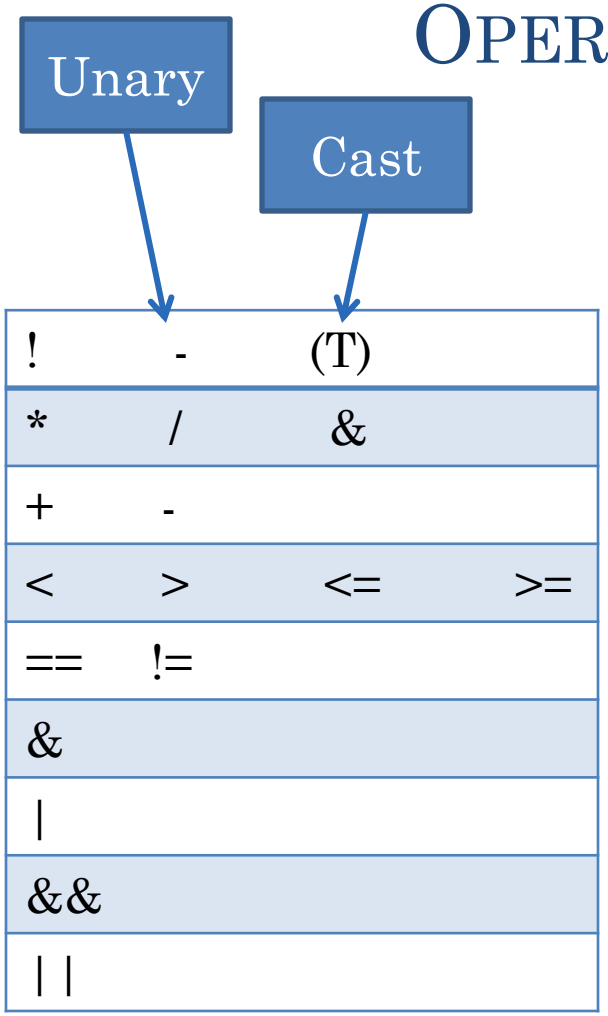
BOOLEAN PROPERTY CODE

```
// declare in interface
private final double HIGH_BMI = 25;


public boolean isOverWeight() {
        return getBmi() > HIGH_BMI;
}
```

Preventing Invalid BMI



Operator Precedence




```
false && 10 / 0 > 0
```

```
- 5 - 4
```

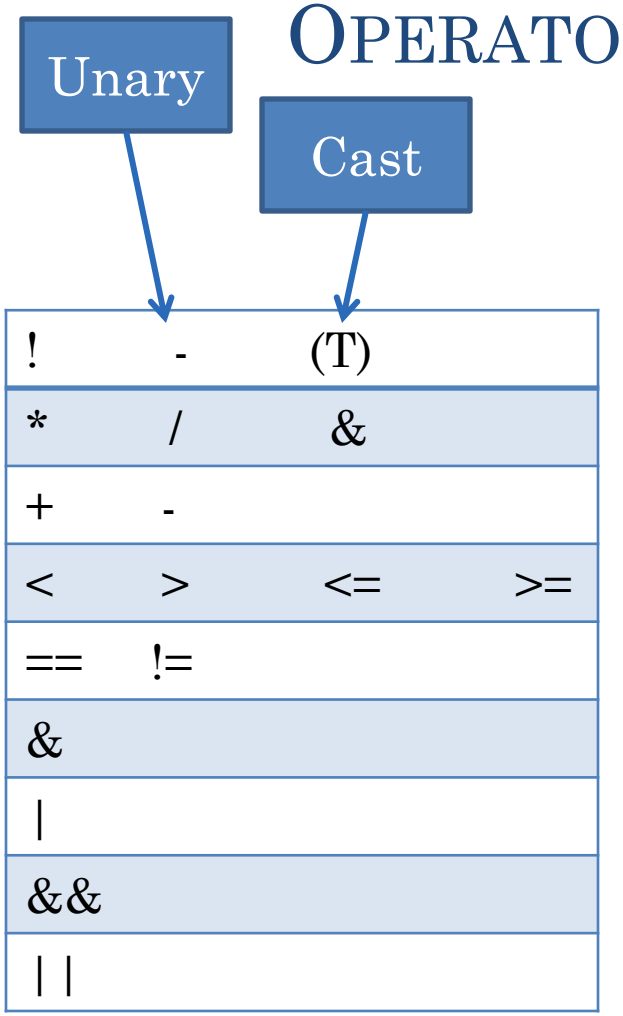
```
!true && false
```

```
5 / 4 * 3
```

```
true || false == false || true
```

```
(int) 5 / 2.0
```

Operator Precedence (Edit)




```
false && 10 / 0 > 0
```

```
- 5 - 4
```

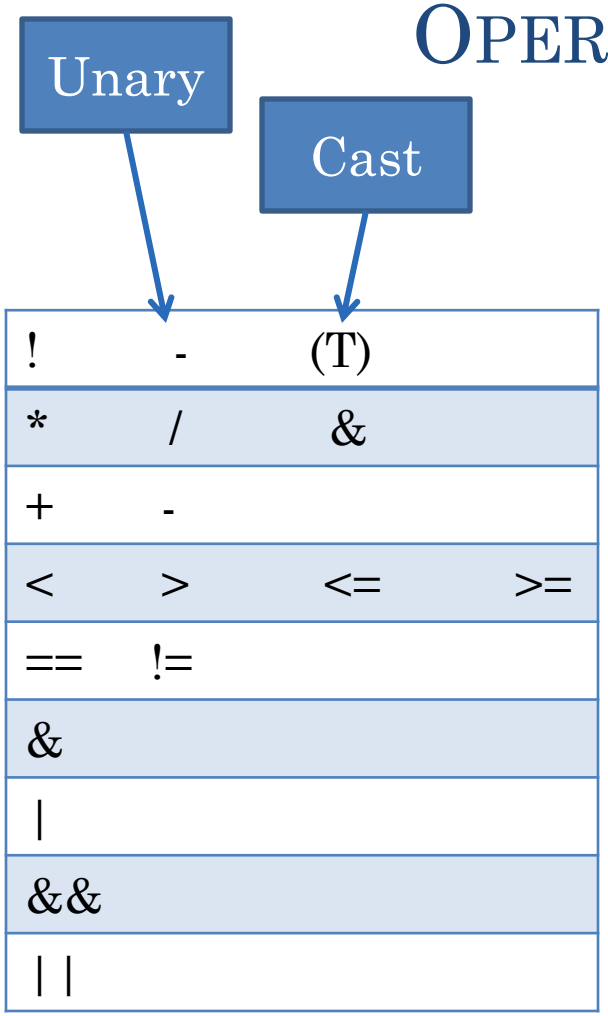
```
!true && false
```

```
5 / 4 * 3
```

```
true || false == false || true
```

```
(int) 5 / 2.0
```

Operator Precedence



false && 10 / 0> 0	→ false && ((10 / 0) > 0))
- 5 - 4	→ (- 5) - 4
!true && false	→ (!true) && false
5 / 4 * 3	→ (5 / 4) * 3
true || false == false || true	→true || (false == false) || true
(int) 5 / 2.0	→ (**(int)** 5) / 2.0

Printing Arbitrary Expressions

System.out.println (2)	Output: 2
System.out.println (2.0)	Output: 2.0
System.out.println ((int) 2.0)	Output: 2
System.out.println (5 > 0)	Output: true