1.2 Built-in Types of Data

Built-in Data Types

Data type. A set of values and operations defined on those values.

type	set of values	literal values	operations
`char`	characters	`'A'` `'@'`	compare

`String`	sequences of characters	`"Hello World"`
		`"126 is fun"`
			concatenate

`int`	integers	`17`
		`12345`
			add, subtract, multiply, divide

`double`	floating-point numbers	`3.1415`
		`6.022e23`
			add, subtract, multiply, divide

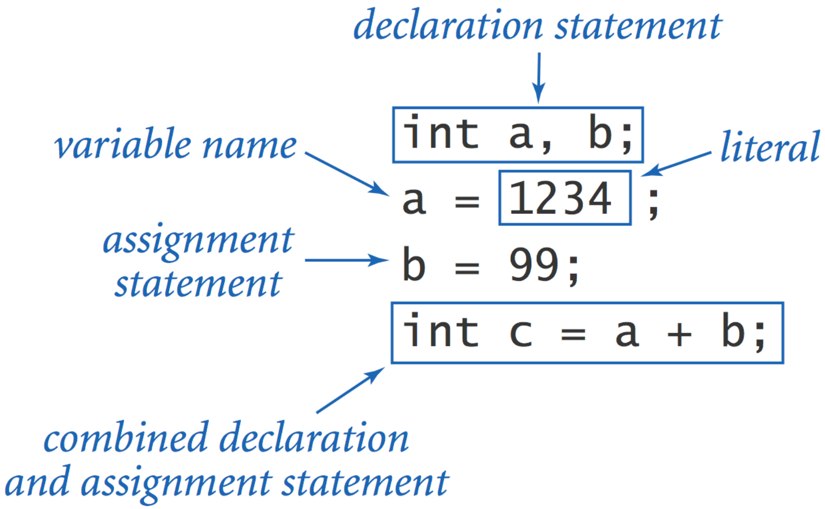
`boolean`	truth values	`true` `false`	and, or, not

Basic Definitions

Variable. A name that refers to a value of declared type.

Literal. Programming language representation of a value.

Assignment statement. Associates a value with a variable.



declaration statement

variable name

literal

assignment statement

```
int a, b;
a = 1234 ;
b = 99;
int c = a + b;
```

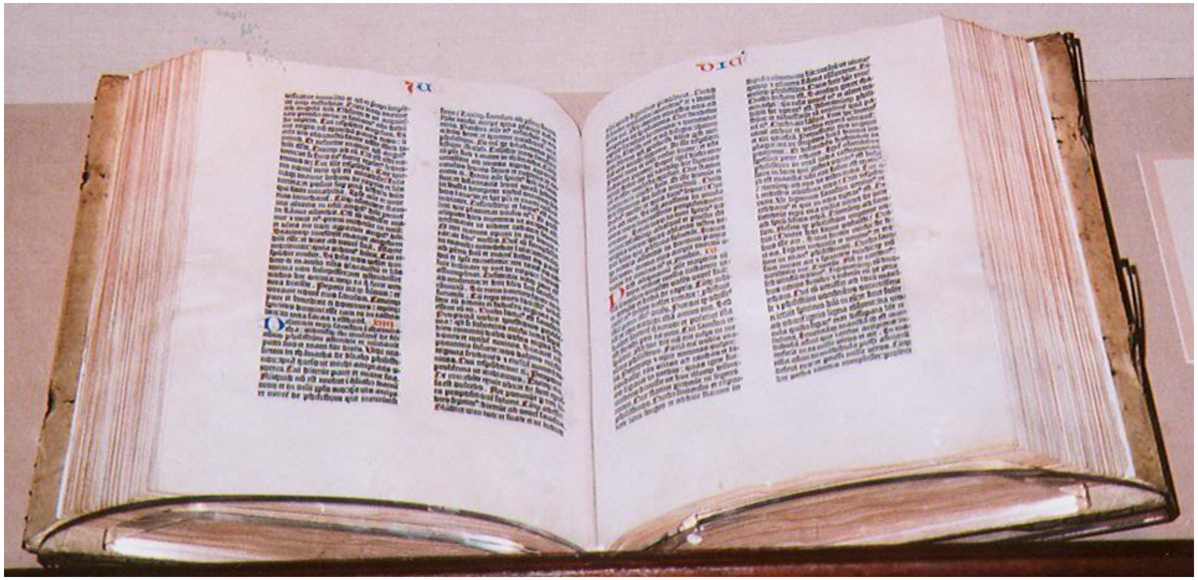
combined declaration and assignment statement

Trace

Trace. Table of variable values after each statement.

	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Text



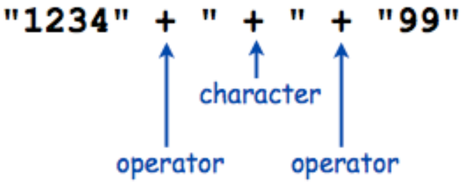
Text

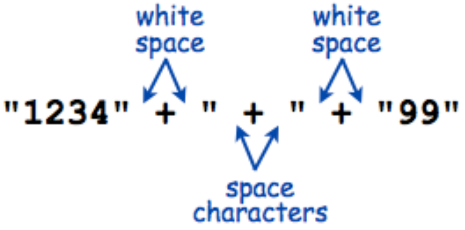
String data type. Useful for program input and output.

values	sequences of characters
typical literals	`"Hello,"  "1 "  " * "`
operation	concatenate
operator	+

expression	*value*
`"Hi, " + "Bob"`	`"Hi, Bob"`
`"1" + " 2 " + "1"`	`"1 2 1"`
`"1234" + " + " + "99"`	`"1234 + 99"`
`"1234" + "99"`	`"123499"`

Caveat. Meaning of characters depends on context.





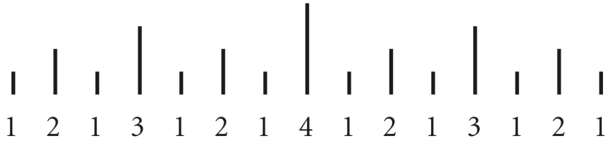
Subdivisions of a Ruler

```
public class Ruler {
   public static void main(String[] args) {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      String ruler4 = ruler3 + " 4 " + ruler3;
      System.out.println(ruler4);
   }
}
```

```
      "1"
    "1 2 1"
"1 2 1 3 1 2 1"
```

string concatenation

```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```



Integers

…, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, …

Integers

int data type. Useful for expressing algorithms.

values	integers between -2^{31} and $+2^{31}-1$				
typical literals	1234 99 -99 0 1000000				
operations	add	subtract	multiply	divide	remainder
operators	+	-	*	/	%

expression	*value*	*comment*
5 + 3	8	
5 - 3	2	
5 * 3	15	
5 / 3	1	no fractional part
5 % 3	2	remainder
1 / 0		run-time error
3 * 5 - 2	13	* has precedence
3 + 5 / 2	5	/ has precedence
3 - 5 - 2	-4	left associative
(3 - 5) - 2	-4	better style
3 - (5 - 2)	0	unambiguous

Integer Operations

```
public class IntOps {
   public static void main(String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

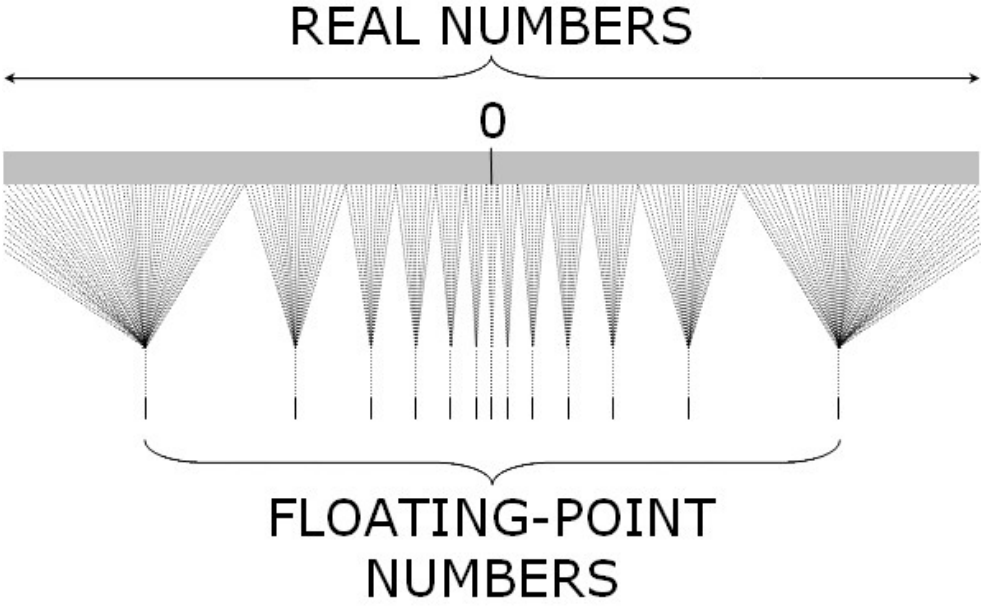
command-line arguments

```
% javac IntOps.java
% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

1234 = 12*99 + 46

Java automatically converts **a**, **b** , and **rem** to type **String**

Floating-Point Numbers



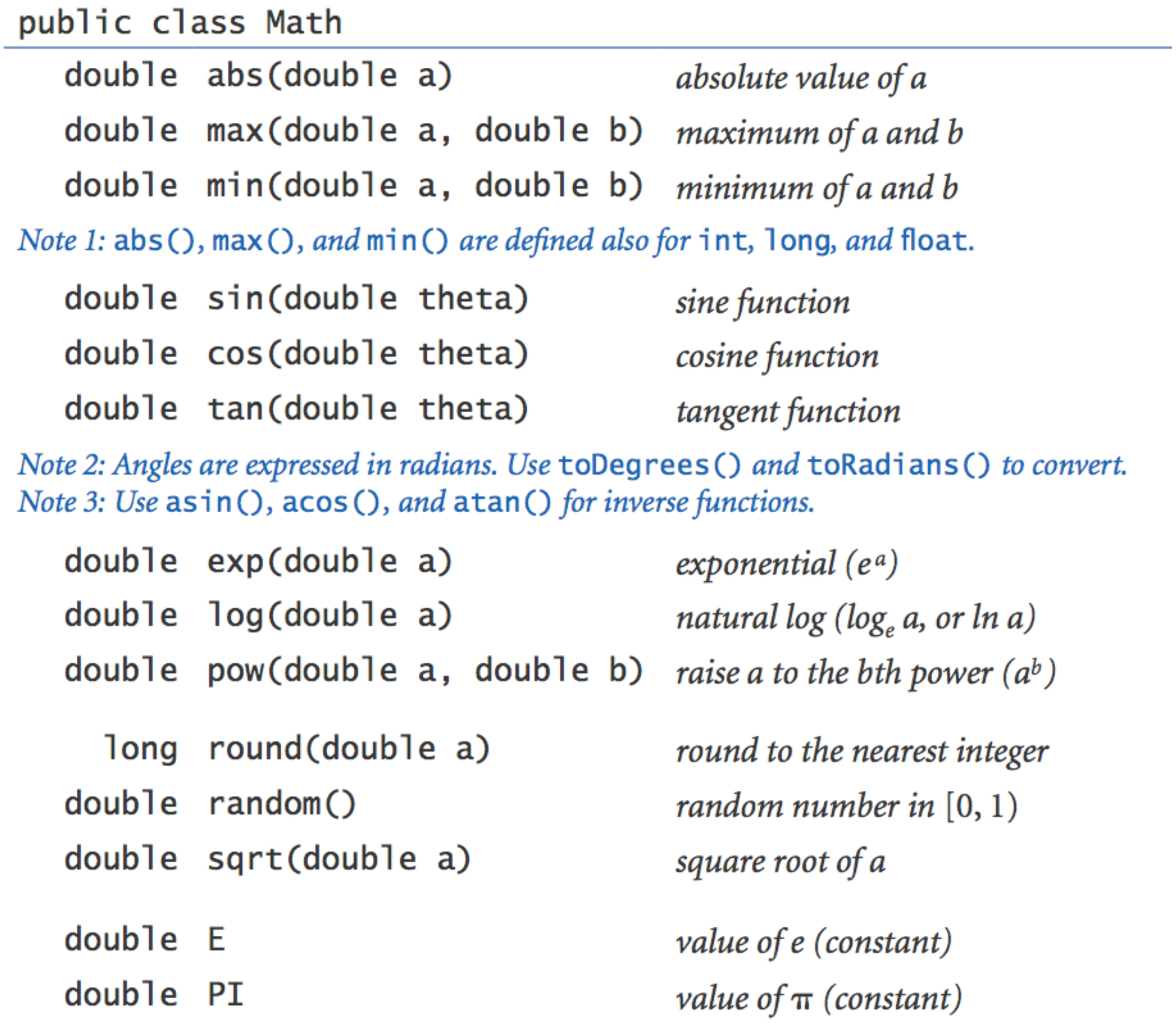
Floating-Point Numbers

`double` data type. Useful in scientific applications.

values	real numbers (specified by IEEE 754 standard)				
typical literals	`3.14159`	`6.022e23`	`-3.0`	`2.0`	`1.4142135623730951`
operations	add	subtract		multiply	divide
operators	`+`	`-`		`*`	`/`

expression	*value*
`3.141 + .03`	`3.171`
`3.141 - .03`	`3.111`
`6.02e23 / 2.0`	`3.01e23`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`1.0 / 0.0`	`Infinity`
`Math.sqrt(2.0)`	`1.4142135623730951`
`Math.sqrt(-1.0)`	`NaN`

Excerpts from Java's Math Library



`public class Math`

`double abs(double a)`	*absolute value of a*
`double max(double a, double b)`	*maximum of a and b*
`double min(double a, double b)`	*minimum of a and b*

Note 1: `abs()`, `max()`, *and* `min()` *are defined also for* `int`, `long`, *and* `float`.

`double sin(double theta)`	*sine function*
`double cos(double theta)`	*cosine function*
`double tan(double theta)`	*tangent function*

Note 2: Angles are expressed in radians. Use `toDegrees()` *and* `toRadians()` *to convert.*
Note 3: Use `asin()`, `acos()`, *and* `atan()` *for inverse functions.*

`double exp(double a)`	*exponential (e^a)*
`double log(double a)`	*natural log ($\log_e a$, or ln a)*
`double pow(double a, double b)`	*raise a to the bth power (a^b)*
`long round(double a)`	*round to the nearest integer*
`double random()`	*random number in [0, 1)*
`double sqrt(double a)`	*square root of a*
`double E`	*value of e (constant)*
`double PI`	*value of π (constant)*

http://download.oracle.com/javase/6/docs/api/java/lang/Math.html

Quadratic Equation

Ex. Solve quadratic equation $x^2 + bx + c = 0$.

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```java
public class Quadratic {
   public static void main(String[] args) {

      // parse coefficients from command-line
      double b = Double.parseDouble(args[0]);
      double c = Double.parseDouble(args[1]);

      // calculate roots
      double discriminant = b*b - 4.0*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / 2.0;
      double root2 = (-b - d) / 2.0;

      // print them out
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

Testing

Testing. Some valid and invalid inputs.

```
% java Quadratic -3.0 2.0
2.0
1.0
```

command-line arguments

$x^2 - 3x + 2$

```
% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949
```

golden ratio

$x^2 - x - 1$

```
% java Quadratic 1.0 1.0
NaN
NaN
```

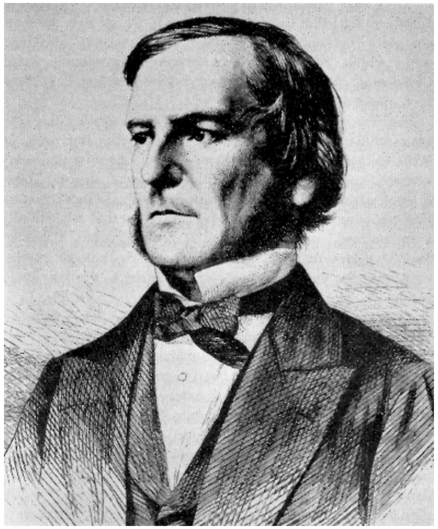
not a number

$x^2 + x + 1$

```
% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello

% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException
```

Booleans



Booleans

boolean data type. Useful to control logic and flow of a program.

values	true or false		
literals	true false		
operations	and	or	not
operators	&&	||	!

a	!a
true	false
false	true

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Comparisons

Comparisons. Take two operands of one type (e.g., `int`) and produce a result of type `boolean`.

op	*meaning*	`true`	`false`
`==`	*equal*	`2 == 2`	`2 == 3`
`!=`	*not equal*	`3 != 2`	`2 != 2`
`<`	*less than*	`2 < 13`	`2 < 2`
`<=`	*less than or equal*	`2 <= 2`	`3 <= 2`
`>`	*greater than*	`13 > 2`	`2 > 13`
`>=`	*greater than or equal*	`3 >= 2`	`2 >= 3`

non-negative discriminant?	`(b*b - 4.0*a*c) >= 0.0`
beginning of a century?	`(year % 100) == 0`
legal month?	`(month >= 1) && (month <= 12)`

Leap Year

Q. Is a given year a leap year?

A. Yes if either (i) divisible by 400 or (ii) divisible by 4 but not 100.

```
public class LeapYear {
   public static void main(String[] args) {
      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;

      // divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);

      // or divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);

      System.out.println(isLeapYear);
   }
}
```

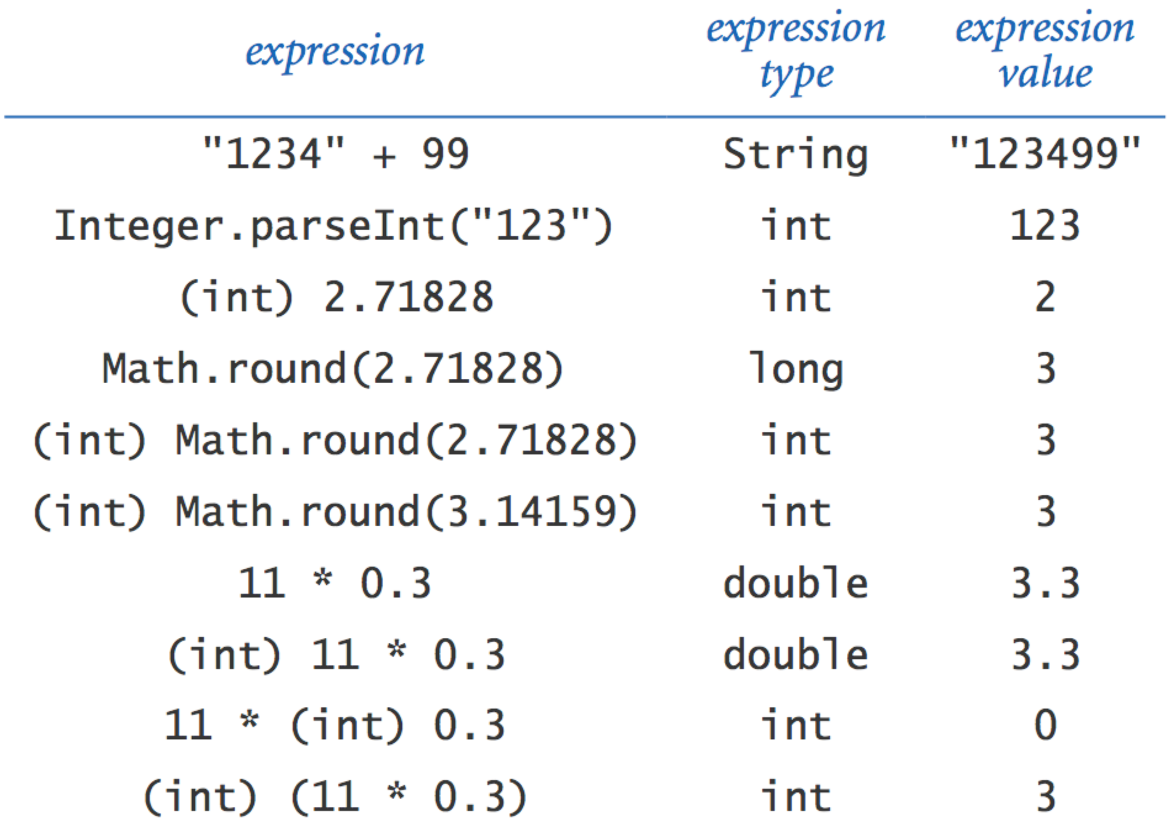
```
% java LeapYear 2004
true
% java LeapYear 1900
false
% java LeapYear 2000
true
```

Type Conversion

Type Conversion

Type conversion. Convert value from one data type to another.

- Automatic: no loss of precision; or with strings.
- Explicit: cast; or method.



expression	*expression type*	*expression value*
`"1234" + 99`	`String`	`"123499"`
`Integer.parseInt("123")`	`int`	`123`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`(int) Math.round(3.14159)`	`int`	`3`
`11 * 0.3`	`double`	`3.3`
`(int) 11 * 0.3`	`double`	`3.3`
`11 * (int) 0.3`	`int`	`0`
`(int) (11 * 0.3)`	`int`	`3`

Random Integer

Ex. Generate a pseudo-random number between 0 and N-1.

```
public class RandomInt {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      double r = Math.random();
      int n = (int) (r * N);

      System.out.println("random integer is " + n);
   }
}
```

String to int (method)

double between 0.0 and 1.0

double to int (cast)

int to double (automatic)

int to String (automatic)

```
% java RandomInt 6
random integer is 3
% java RandomInt 6
random integer is 0
% java RandomInt 10000
random integer is 3184
```

Summary

A data type is a set of values and operations on those values.

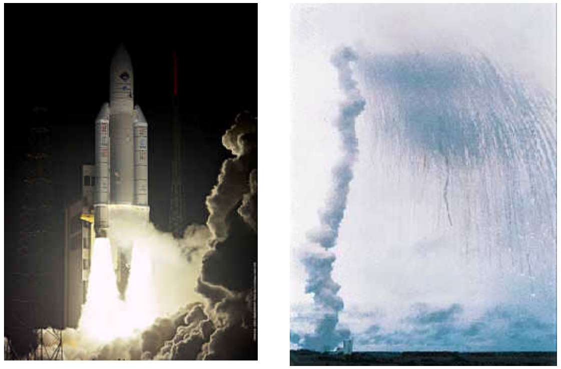
- `String` text processing.
- `double,int` mathematical calculation.
- `boolean` decision making.

In Java, you must:

- Declare type of values.
- Convert between types when necessary.

Why do we need types?

- Type conversion must be done at some level.
- Compiler can help do it correctly.
- Ex 1: in 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion.
- Ex 2: `i = 0` in Matlab redefines $\sqrt{-1}$.



example of bad type conversion

Assignments #1

- Due: before next lab class (before Wed. 9:30)
- Ex 1.2.3, 1.2.7, 1.2.11, 1.2.15, 1.2.29
- How to submit:

1. You prepare your HW by using wordprocesses (e.g. Word, HWP…)
   Note: Don't use papers!
2. When you need to write code, make the separated files. (and refer to the files in the document.)
3. Zip all files
4. Submit the zib file to:
   https://www.dropbox.com/request/UWMpnJ5ojs9fn1qkRiW2

5. Important!: Input your name correctly!

Extra Slides

Initializing Variables

Q. What happens if I forget to initialize the variable `a` or `b`?

- Java compiler does not allow this.
- Caveat: in other languages, variable initialized to arbitrary value.

	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234