INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

Any Questions ?

2. Algorithm Implementation and Counting

https://introcs.cs.rutgers.edu

Before we get started

Are there any questions about Assignment Directions?

- Provide
  a) inputs
  b) outputs
  c) error conditions or pre-conditions
  d) an algorithm using the pseudocode reference guide used in class
  e) a set of test cases
  f) the minimum number of operations executed in your algorithm
  g) the maximum number of operations executed in your algorithm

- https://introcs.cs.rutgers.edu/pseudocode-a1-spring2022/

Example

Problem. Ellen and Ana are moving together. Determine how many cats they will have in total?

```
READ ellensCats
READ anasCats
IF ellensCats < 0 OR anasCats < 0 THEN
    DISPLAY ERROR
ELSE
    COMPUTE totalCats AS ellensCats + anasCats
    DISPLAY totalCats
ENDIF
```

Test Cases:

[2,3] [5]
[0,3] [3]
[3,0] [3]
[-1,3] [ERROR]
[3,-1] [ERROR]

a) Inputs: ***Two integer values: Ana's # cats, Ellen's # cats***
b) Outputs: ***One integer value: Total number of cats***
c) error conditions or pre-conditions
Error Conditions: Ana's cats < 0, Ellen's cats < 0 (either input < 0)
d) an algorithm using the pseudocode reference used in class--***on left***
e) a set of test cases –***on left***
f) the minimum number of operations executed in your algorithm ***5***
g) the maximum number of operations executed in your algorithm ***6***

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

3. Programming in Java

https://introcs.cs.rutgers.edu

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational operators

Adopted and modified from Slides by Sedgewick and Wayne

Our Choice: Java

Java features

- Widely used.
- Widely available.
- Continuously under development since early 1990s.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

Facts of life

- No language is perfect.
- You need to start with *some* language.

> *“ There are only two kinds of programming languages: those people always [gripe] about and those nobody uses.”*
>
> *– Bjarne Stroustrup*

Our approach

- **Use a minimal subset of Java.**
- **Develop general programming skills that are applicable to many languages.**

It’s not about the language!

A rich subset of the Java language vocabulary—WE ARE TAKING THIS SLOWLY!!!

built-in types
int
long
double
char
String
boolean

operations on numeric types
+
-
*
/
%
++
--

String *operations*
+
""
length()
charAt()
compareTo()
matches()

assignment
=

flow control
if
else
for
while

Object oriented
static
class
public
private
new
final
toString()
main()

Math *methods*
Math.sin()
Math.cos()
Math.log()
Math.exp()
Math.pow()
Math.sqrt()
Math.min()
Math.max()
Math.abs()
Math.PI

punctuation
{
}
(
)
,
;

comparisons
<
<=
>
>=
==
!=

boolean operations
true
false
!
&&
||

arrays
a[]
length
new

type conversion methods
Integer.parseInt()
Double.parseDouble()

System *methods*
System.print()
System.println()
System.printf()

our Std methods
StdIn.read*()
StdOut.print*()
StdDraw.*()
StdAudio.*()
StdRandom.*()

Your programs will primarily consist of these plus identifiers (names) that you make up.

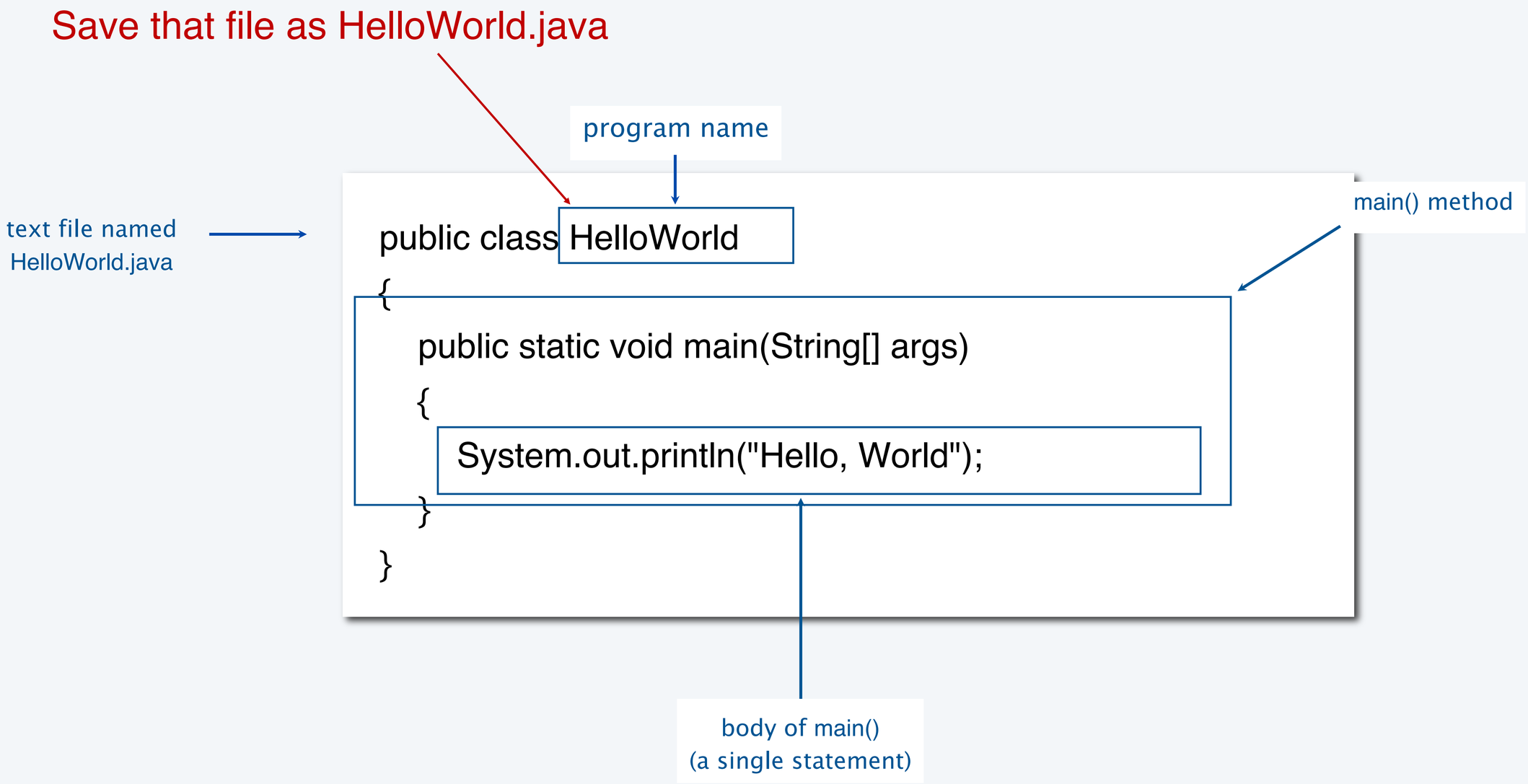
COMPUTER ORGANIZATION-SUGGESTION

- Create a folder ON YOUR DESKTOP to hold your Java Programs
    - Easy to navigate to!
- OPEN VISUAL STUDIO CODE
    - FILE→OPEN (NAVIGATE TO THAT FOLDER)
    - OPEN
    - File Menu: NEW FILE– (Choose JAVA if asked for a language)

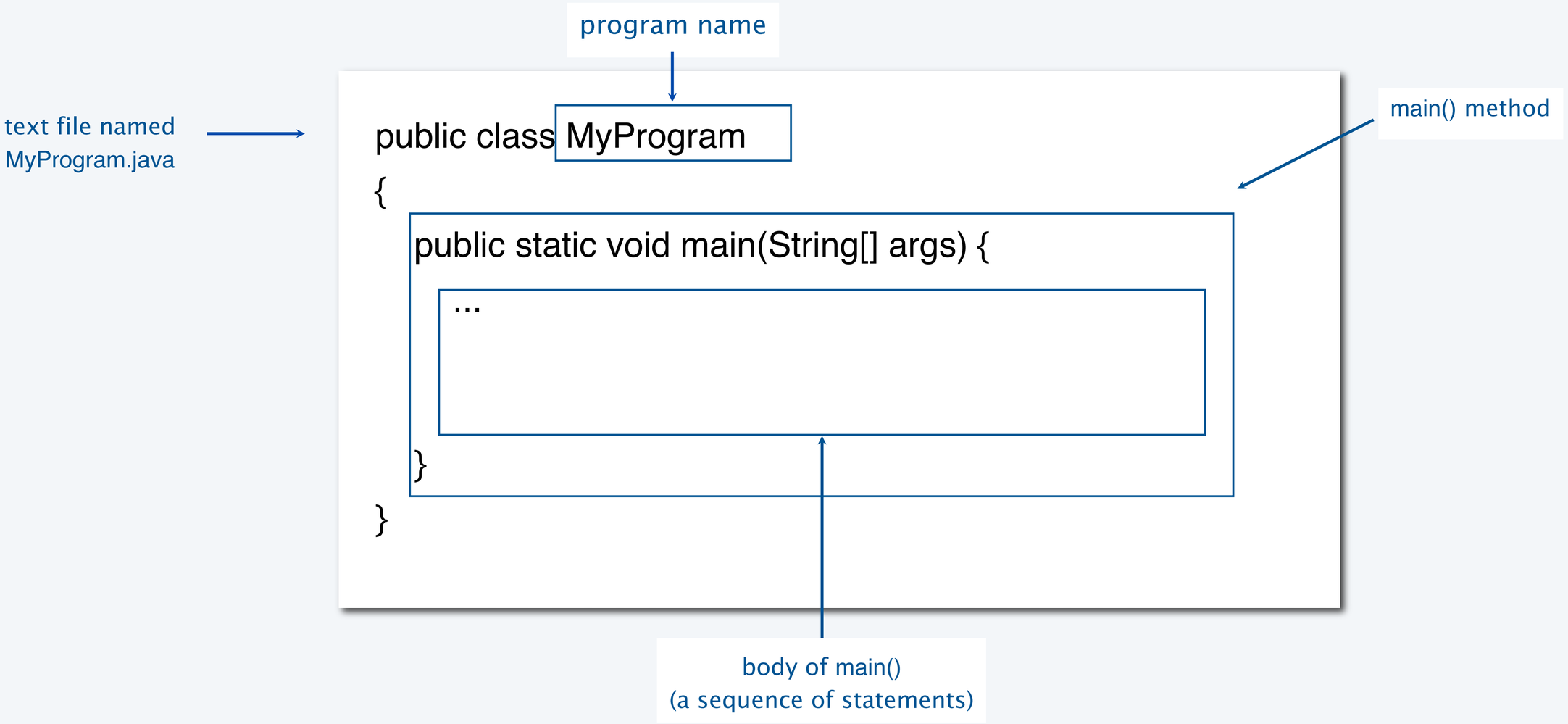
Terminal commands

- Macs: https://github.com/0nn0/terminal-mac-cheatsheet
- Windows: https://serverspace.us/support/help/windows-cmd-commands-cheat-sheet/

Anatomy of your first program



Anatomy of your next several programs



What does it take to run a Java program?

Java program also called *source code*

Compile

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

HelloWorld.java

input → Compiler → output

The compiler is a program that takes as an input the *source code* and outputs the *byte code.*

To compile:
javac HelloWorld.java

Java program in *ByteCode*

Your Java program translated into a language that is suitable for execution on the computer. Very difficult for humans to read and understand.

HelloWorld.class

What does it take to run a Java program?

Compile

Java program also called *source code*

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

HelloWorld.java

input → Compiler → output

The compiler is a program that takes as an input the *source code* and outputs the *byte code.*

To compile:
javac HelloWorld.java

Java program in *ByteCode*

Your Java program translated into a language that is suitable for execution on the computer. Very difficult for humans to read and understand.

HelloWorld.class

Execute

Java program in *ByteCode*

Your Java program translated into a language that is suitable for execution on the computer. Very difficult for humans to read and understand.

HelloWorld.class

input → Interpreter: the Java Virtual Machine

The interpreter is a program that takes as an input the *byte code* and executes the instructions on the computer.

To execute:
java HelloWorld

Pop quiz on "your first program"

Q. Use common sense to cope with the following error messages.

```
% javac MyProgram.java
% java MyProgram
Main method not public.
```

```
% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
      public static main(String[] args)
                    ^
```

Pop quiz on "your first program"

Q. Use common sense to cope with the following error messages.

```
% javac MyProgram.java
% java MyProgram
Main method not public.
```

A. Must have forgotten “public”.

public static void main(String[] args)

```
% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
      public static main(String[] args)
                    ^
```

A. Check HelloWorld. Aha! Forgot “void”.

public static void main(String[] args)

Three versions of the same program.


```
public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Hello, World");
   }
}
```

Not very exciting

Descriptive comments

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *************************************************************************/

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

REALLY BAD!

```
public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }
```

Lesson: Fonts, color, comments, and extra space are not relevant in Java language.

Note on program style

Different styles are appropriate in different contexts.

- Integrated development environment
- Booksite
- Book
- Your code

Enforcing consistent style can

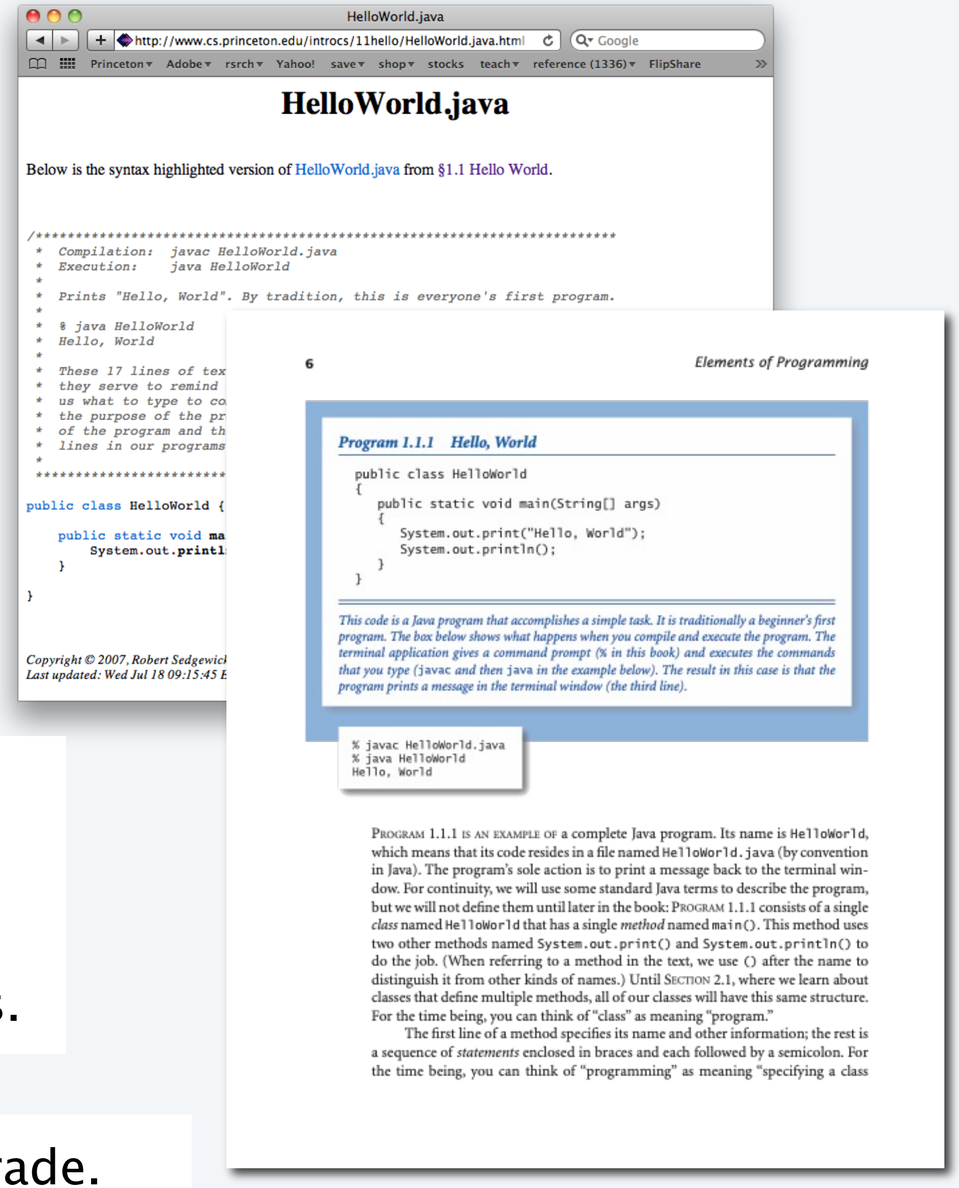
- Stifle creativity.
- Confuse style with language.

Emphasizing consistent style can

- Make it easier to spot errors.
- Make it easier for others to read and use code.
- Enable development environment to provide visual cues.

Bottom line for you: Listen to the person assigning your grade.

or your boss!



Program development in Java

is a three-step process, *with feedback*

1. EDIT your program
   - Create it by typing on your computer's keyboard.

   Result: a text file such as HelloWorld.java.

2. COMPILE it to create an executable file
   - Use the Java compiler
   - Result: a Java bytecode file such as HelloWorld.class

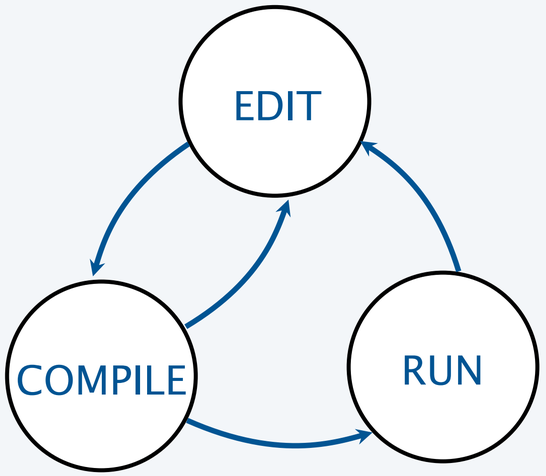
   not a legal Java program

   - Mistake? Go back to 1. to fix and recompile.

3. RUN your program
   - Use the Java runtime.
   - Result: your program's output.
   - Mistake? Go back to 1. to fix, recompile, and run.

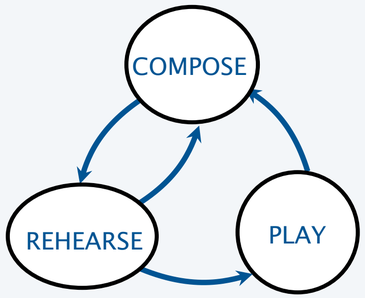
   a legal Java program that does the wrong thing

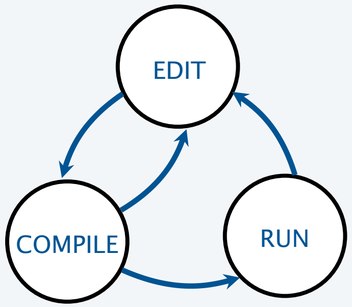
SAVE



Software for program development

Any creative process involves cyclic refinement/development.





A significant difference with programs: *We can use our computers to facilitate the process.*

Program development environment: Software for editing, compiling and running programs.

Two time-tested options: (Stay tuned for details).

Virtual terminals

- Same for many languages and systems.
- Effective even for beginners.

Bottom line: Extremely simple and concise.

Integrated development environment

- Often language- or system-specific.
- Can be helpful to beginners.

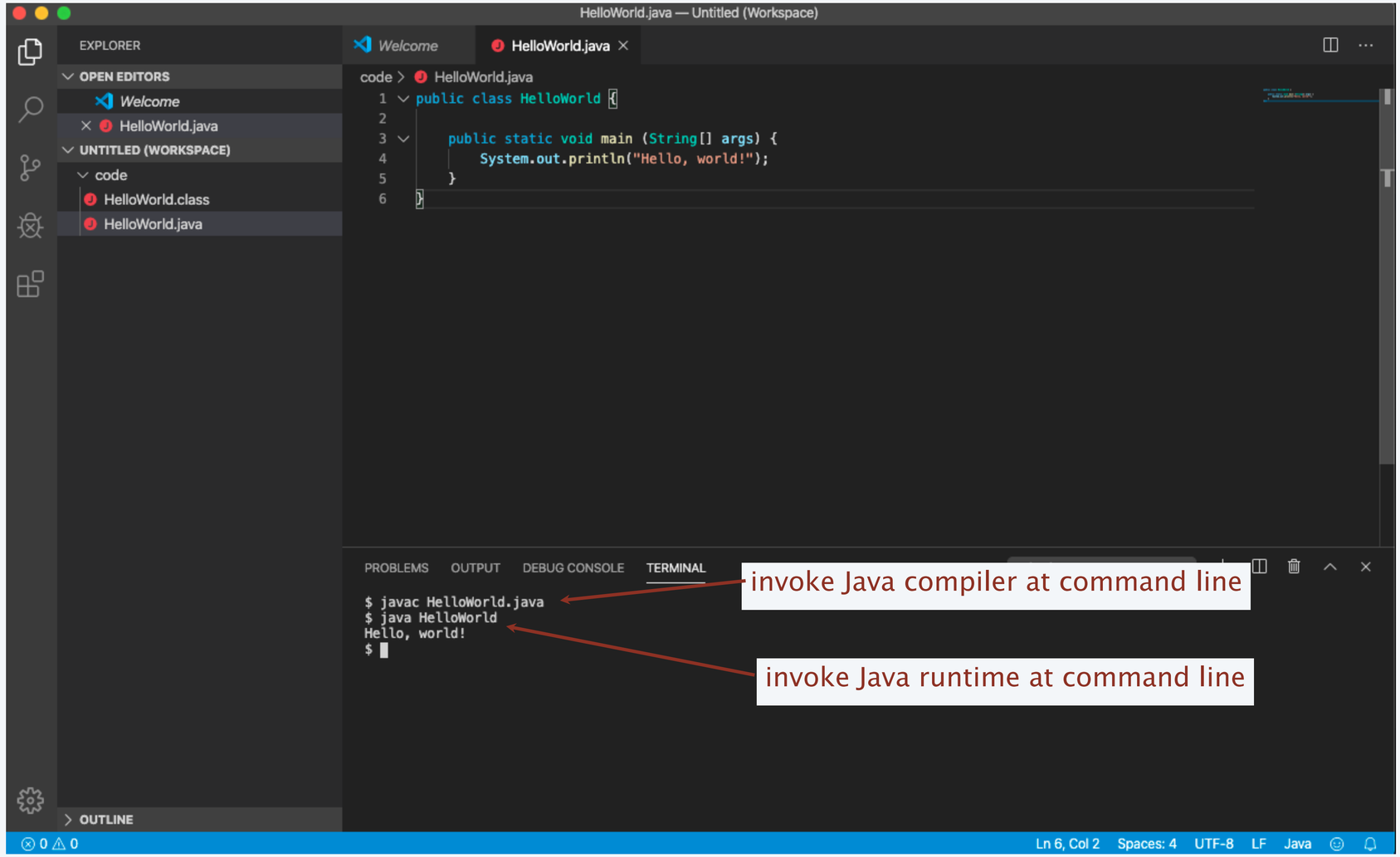
Bottom line: Variety of useful tools.

Program development (In this course)

Use a text editor for code: **Visual Studio Code**

Command line to compile and execute

Installation guide can be found at Week 3 resources on **introcs.cs.rutgers.edu/lectures**



3. Programming in Java

- Java Programs
- **Built-in data types**
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational operators

Built-in data types

LO 3.2a

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

type	*set of values*	*examples of values*	*examples of operations*
char	characters	'A' '@'	compare
String	sequences of characters	"Hello World" "CS 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

Java's built-in data types

Primitive Data Types in Java

Commonly used Java primitive types that we use in this course:

Type	Description	Examples
`int`	**Integers**	`34, 5, -6, 20567, 0`
`double`	**Real numbers**	`1.2, 4.0, -19.45`
`char`	**Single characters**	`'a', 's', '-', '2'`
`boolean`	**Logical values**	`true, false`

Pop quiz on data types

Q. What is a data type?

A. A set of values and a set of operations on those values.

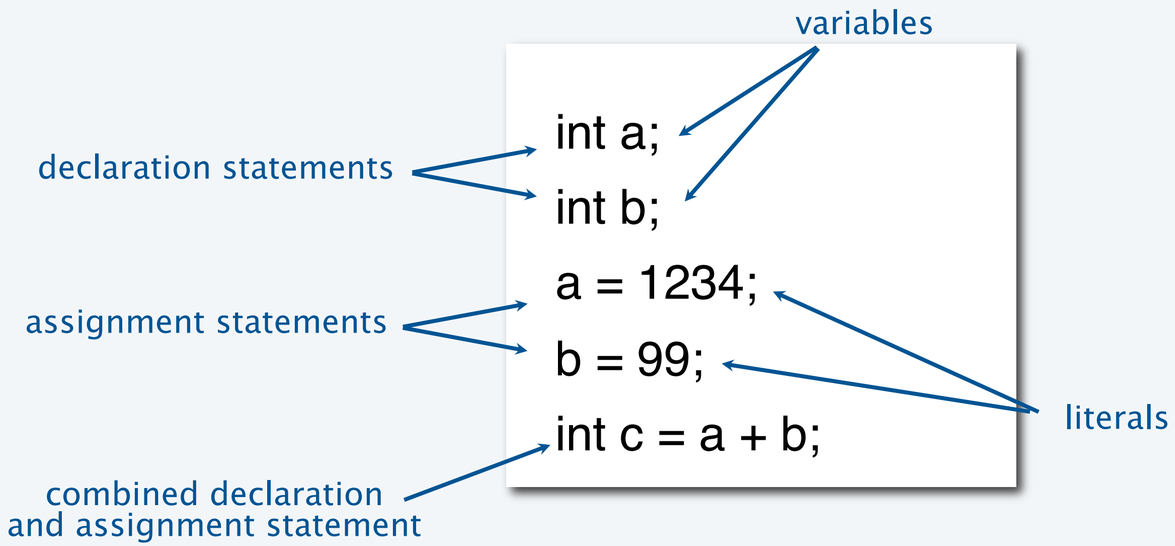
Basic Definitions

A variable is a name that refers to a value.

A literal is a programming-language representation of a value.

A declaration statement associates a variable with a type.

An assignment statement associates a value with a variable.



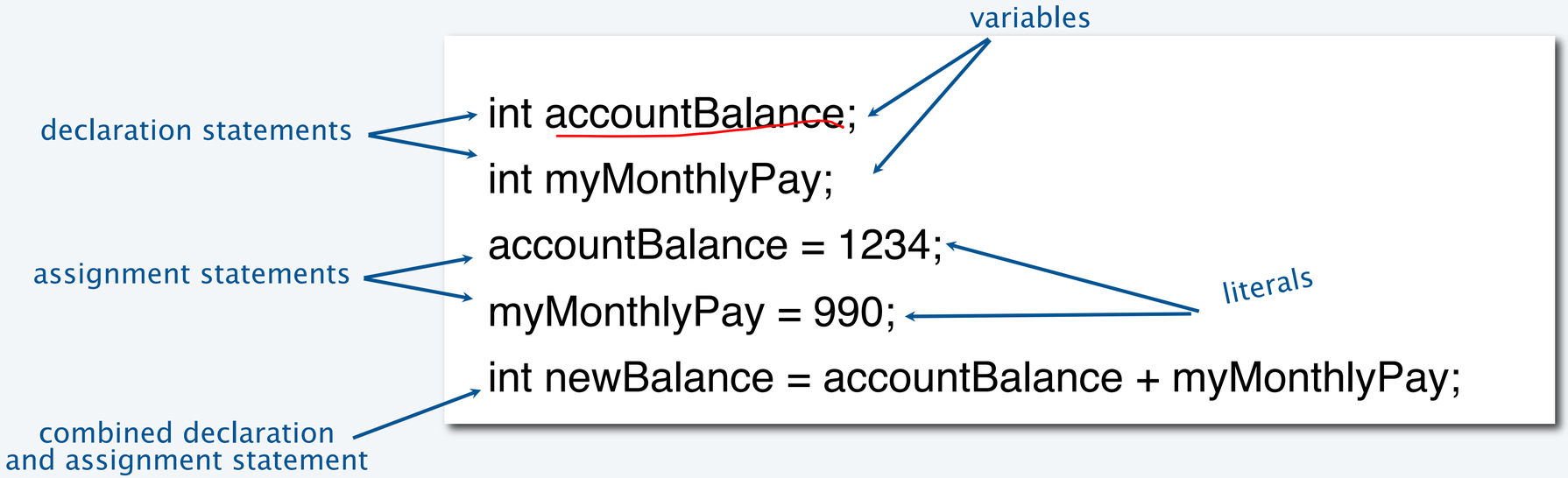
Naming variable conventions

A variable is a name that refers to a value.

The name of a variable must reflect what the variable does.

A proper style of choosing a name make it easy to read your code.

We use Camel Case style for naming variables.



Camel case is the practice of writing phrases such that each word or abbreviation in the middle of the phrase begins with a capital letter - Wikipedia

Assignment involving arithmetic operations


```
public class Assign
{  public static void main(String[] args)
   {
      int a = 1234;
      int b = 99;
      int t = a;
      a = a + b;
      b = a - t;
   }
}
```

A trace is a table of variable values after each statement.

	a	b	t
	undeclared	*undeclared*	*undeclared*
int a = 1234;	1234	*undeclared*	*undeclared*
int b = 99;	1234	99	*undeclared*
int t = a;	1234	99	1234
a = a + b;	1333	99	1234
b = a - t;	1333	99	1234

Q. What are the final values of a, b, and t?

A. Trace the program to find the values of a, b and t. a= 1333, b=99, t = 1234

Data type for computing with strings: String

LO 3.2f

String data type

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

Examples of String operations (concatenation)

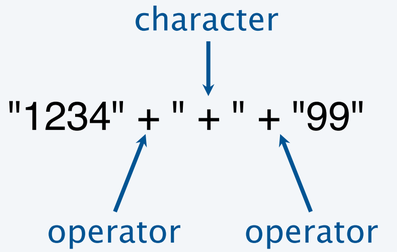
expression	*value*
"Hi, " + "Bob"	"Hi, Bob"
"1" + " 2 " + "1"	"1 2 1"
"1234" + " + " + "99"	"1234 + 99"
"1234" + "99"	"123499"

Typical use: Input and output.

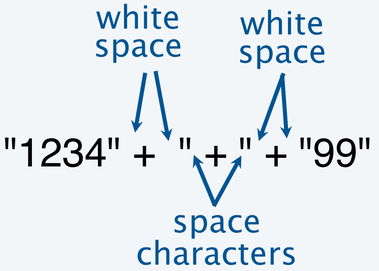
Important note:

Character interpretation depends on context!

Ex 1: plus signs



Ex 2: spaces



Example of computing with strings: subdivisions of a ruler

LO 3.2f

```
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);
   }
}
```

all + ops are concatenation

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

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

	ruler1	ruler2	ruler3	ruler4
	undeclared	*undeclared*	*undeclared*	*undeclared*
ruler1 = "1";	1	*undeclared*	*undeclared*	*undeclared*
ruler2 = ruler1 + " 2 " + ruler1;	1	1 2 1	*undeclared*	*undeclared*
ruler3 = ruler2 + " 3 " + ruler2;	1	1 2 1	1 2 1 3 1 2 1	*undeclared*
ruler4 = ruler3 + " 4 " + ruler3;				1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Data type for computing with integers: int

int data type

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

Important note:

Only 2^{32} different int values.

not *quite* the same as integers

Examples of int operations

expression	*value*	*comment*
5 + 3	8	
5 - 3	2	
5 * 3	15	
5 / 3	1	*drop fractional part*
5 % 3	2	*remainder*
1 / 0		*runtime error*

Precedence

expression	*value*	*comment*
3 * 5 - 2	13	** has precedence*
3 + 5 / 2	5	*/ has precedence*
3 - 5 - 2	-4	*left associative*
(3 - 5) - 2	-4	*better style*

Typical usage: Math calculations; specifying programs (stay tuned).

Example of computing with integers and strings, with type conversion

LO 3.2d, 3.3abc

```
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);
   }
}
```

Java automatically converts int values to String for concatenation

- Go to Course Website
- Click on Book Site
- (left menu)→Click on Web Resources→Code
- Click on the program you want.
- Select the code; COPY
- Paste in NEW FILE in VSC

Let's try this!

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

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

Note: 1234 = 12*99 + 46

Ana's Cats

```
READ anasCats
READ ellensCats
IF ellensCats < 0 OR anasCats < 0 THEN
    DISPLAY ERROR
ELSE
    COMPUTE totalCats AS ellensCats + anasCats
    DISPLAY totalCats
ENDIF
```

```
public class Cats{
   public static void main(String[] args) {
      int anaCats = Integer.parseInt(args[0]);
      int ellenCats = Integer.parseInt(args[1]);
      if (anaCats < 0 || ellenCats < 0) {
        System.out.println("error");
      }
      else {
          int totalCats = anaCats + ellenCats;
          System.out.println(totalCats);
      }
   }
}
```

Pay for Hours Worked

ALGORITHM:

```
READ hoursWorked
READ ratePerHour
IF hoursWorked < 0 OR ratePerHour < 0 THEN
    DISPLAY ERROR
ELSE
    COMPUTE totalPay  AS hoursWorked *
ratePerHour
    DISPLAY totalPay
ENDIF
```

```
public class Pay{
  public static void main(String args[]){
    double hoursWorked=Double.parseDouble(args[0]);
    double ratePerHour=Double.parseDouble(args[1]);
    double totalPay;
    if (hoursWorked < 0 || ratePerHour < 0) {
        System.out.println("Error");
    }
   else {
        totalPay = hoursWorked * ratePerHour;
        System.out.println(totalPay);
    }
  }
}
```

Temps –Your turn

Solution Design. Input. A temperature in Fahrenheit (double)

- Output. Equivalent Celsius
- Error. Temperature below absolute zero (-459.67 degrees F)

ALGORITHM:

READ tempF

IF tempF < -459.67 **THEN**

DISPLAY ERROR: Temperature must be greater than or equal to -459.67

ELSE

COMPUTE tempC **AS** (tempF – 32) * 5/9

DISPLAY tempC

ENDIF

Evens Your turn

INPUT: an integer

OUTPUT: odd or even

ERROR CONDITIONS: none

ALGORITHM:

```
READ number
IF  number %  2 IS  0 THEN
    DISPLAY even
ELSE
   DISPLAY odd
ENDIF
```

Data type for computing with floating point numbers: double

double data type

values	real numbers				
typical literals	3.14159 2.0 1.4142135623730951 6.022e23				
operations	add	subtract	multiply	divide	remainder
operator	+	–	*	/	%

6.022×10^{23}

Typical double values are *approximations*

Examples:

no double value for π.

no double value for $\sqrt{2}$

no double value for 1/3.

Examples of double operations

expression	*value*
3.141 + .03	3.171
3.141 - .03	3.111
6.02e23/2	3.01e23
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
Math.sqrt(2.0)	1.4142135623730951

Special values

expression	*value*
1.0 / 0.0	Infinity
Math.sqrt(-1.0)	NaN

"not a number"

Typical use: Scientific calculations.

Other built-in numeric types – *not too worried about these in this course.*

short data type	
values	integers between -2^{15} and $2^{15}-1$
operations	[same as int]

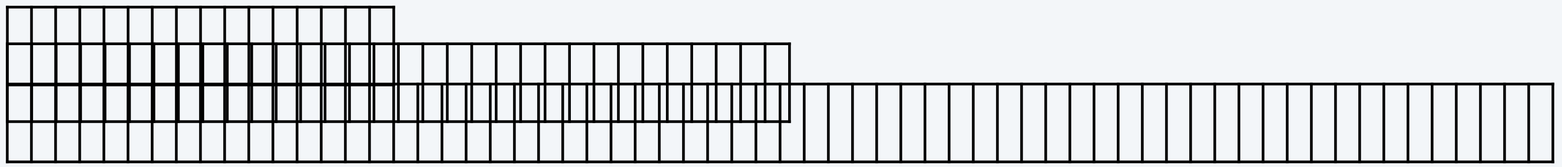
long data type	
values	integers between -2^{63} and $2^{63}-1$
operations	[same as int]

float data type	
values	real numbers
operations	[same as double]

Why different numeric types?

- Tradeoff between memory use and range for integers.
- Tradeoff between memory use and precision for real numbers.

short
int, float
long, double



Compound assignment operators


```
int a = 0;
a = a + 2;
a += 2
```

← Increment the value of a by 2

← Increment the value of a by 2

a = a*3 Equivalent to a *= 3

a = a - 3 Equivalent to a -= 3

a = a / 3 Equivalent to a /= 3

Compound assignment operators

+= -= *= /= %=

Provides a better way to express operations on a single variable.

Example.

Increment variable a by 3

a = a + 3 or
a += 3

The latter is more efficient to compute (one operator += as opposed to two operators (+ and =)

3. Programming in Java

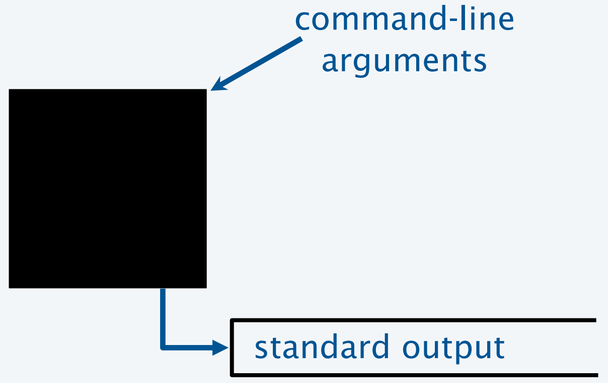
- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational operators

Input and output

is necessary for us to provide data to our programs and to learn the result of computations.

Humans prefer to work with strings.

Programs work more efficiently with numbers.



Bird's eye view of a Java program

Output

- System.out.println() method prints the given string.
- Java automatically converts numbers to strings for output.

Command-line input

- Strings you type after the program name are available as args[0], args[1], ... at *run* time.
- Q. How do we give an *integer* as command-line input?
- A. Need to call System method Integer.parseInt() to convert the strings to integers.

Stay tuned for many more options for input and output, and more details on type conversion.

Input and output warmup: exchange values

LO 3.3abc

```
public class Exchange
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int t = a;
      a = b;
      b = t;
      System.out.println(a);
      System.out.println(b);
   }
}
```

Java automatically converts int values to String for output

```
% java Exchange 5 2
2
5

% java Exchange 1234 99
99
1234
```

Q. What does this program do?

A. Reads two integers from the command line, then prints them out in the opposite order.

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational operators

Data type for computing with true and false: boolean

boolean data type

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

Truth-table definitions

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

Q. a XOR b?

A. (!a && b) || (a && !b)

Proof---next slide!

(!a && b) ll (a && !b)

a	b	!a	!b	!a && b	a && !b	(!a && b) || (a && !b)
T	T					
T	F					
F	T					
F	F					

Data type for computing with true and false: boolean

boolean data type

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

Truth-table definitions

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

Q. a XOR b?

A. (!a && b) || (a && !b)

Proof:

A	b	!a && b	a && !b	(!a && b) || (a && !b)
false	false	false	false	false
false	true	true	false	true
true	false	false	true	true
true	true	false	false	false

Typical usage: Control logic and flow of a program.

Comparison operators

Fundamental operations that are defined for each primitive type allow us to *compare* values.

- Operands: two expressions of the same type.
- Result: a value of type boolean.

operator	*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

Examples

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

Typical double values are *approximations* so beware of == comparisons

DO NOT USE == with doubles!

Example of computing with booleans: leap year test

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 2016
true

% java LeapYear 1993
false

% java LeapYear 1900
false

% java LeapYear 2000
true
```

Evaluating Boolean Expressions with Truth Tables – PRACTICE!!

LO 3.4bc

a	!a		a	b	a && b	a ll b
true	false		false	false	false	false
false	true		false	true	false	true
			true	false	false	true
			true	true	true	true

See worksheet provided: Week 03 – TruthTables.xlsx

x, y, z are Boolean variables. Build a truth table to evaluate each of the following.

a. (x && y) || z
b. x && y && z
c. (x || y) && z
d. !x && !y
e. !(x || y)
f. !x || !y
g. !(x && y)

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The Math class
- Casting
- Relational operators

Excerpts from Java's Math Library https://docs.oracle.com/javase/7/docs/api/

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

also defined for int, long, and float

inverse functions also available: asin(), acos(), and atan()

Degrees in radians. Use toDegrees() and toRadians() to convert.

You can discard your calculator now (please).

Example of computing with floating point numbers: quadratic equation

From algebra: the roots of $x^2 + bx + c$ are $\frac{-b \pm \sqrt{b^2 - 4c}}{2}$

```
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 of x*x + b*x + c.
      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);
   }
}
```

```
% java Quadratic –3.0 2.0
2.0
1.0
```
$x^2 - 3x + 2$

```
% java Quadratic –1.0 –1.0
1.618033988749895
-0.6180339887498949
```
$x^2 - x - 1$

```
% java Quadratic 1.0 1.0
NaN
NaN
```
$x^2 + x + 1$

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

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

Need two arguments.
(Fact of life: Not all error messages are crystal clear.)

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational operators

Type checking

Types of variables involved in data-type operations always must match the definitions.

The Java compiler is your *friend*: it checks for type errors in your code.

```
public class BadCode
{
   public static void main(String[] args)
   {
      String s = "123" * 2;
   }
}
```

```
% javac BadCode.java
BadCode.java:5: operator * cannot be applied to java.lang.String,int
      String s = "123" * 2;
                       ^
1 error
```

When appropriate, we often *convert* a value from one type to another to make types match.

Type conversion with built-in types

Type conversion is an essential aspect of programming.

Automatic

- Convert number to string for "+".
- Make numeric types match if no loss of precision.

expression	*type*	*value*
"x: " + 99	String	"x: 99"
11 * 0.25	double	2.75

Explicitly defined for function call.

Integer.parseInt("123")	int	123
Math.round(2.71828)	long	3

Cast for values that belong to multiple types.

- Ex: small integers can be short, int or long.
- Ex: double values can be truncated to int values.

(int) 2.71828	int	2
(int) Math.round(2.71828)	int	3
11 * (int) 0.25	int	0

Pay attention to the type of your data.

← Type conversion can give counterintuitive results but gets easier to understand with practice

Pop quiz on type conversion

Q. Give the type and value of each of the following expressions.

a. `( 7 / 2 ) * 2.0`

b. `( 7 / 2.0 ) * 2`

c. `"2" + 2`

d. `2.0 + "2"`

Pop quiz on type conversion

Q. Give the type and value of each of the following expressions.

a. `( 7 / 2 ) * 2.0` — 6.0, a double (7/2 is 3, an int)

b. `( 7 / 2.0 ) * 2` — 7.0, a double

c. `"2" + 2` — 22, a String

d. `2.0 + "2"` — 2.02, a String

Pop quiz on explicit casting

Q. Give the type and value of each of the following expressions.

a.	((double) 7 / 2) * 2.0	7.0, a double
b.	(7 / 2.0) * 2	7.0, a double
c.	"2" + (String)2	22, a String
d.	2.0 + "2"	2.02, a String

Pop quiz on explicit casting error

LO 3.6e

Q. What is the difference between (double)(3/5) and (double)3/5

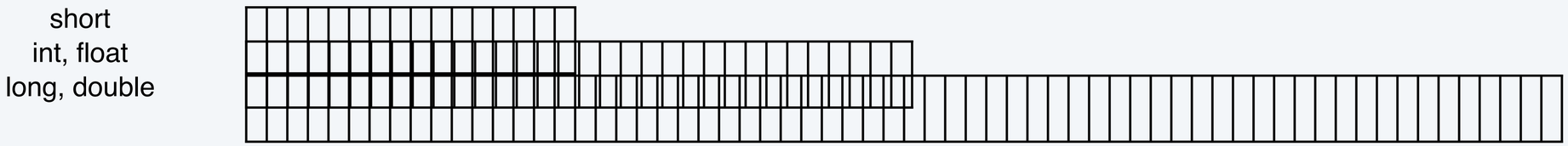
A. (double)(3/5) = 0.0

B. (double)3/5 = 0.6

An instructive story about type conversion

Why different numeric types?

- Tradeoff between memory use and range for integers.
- Tradeoff between memory use and precision for floating-point.



A conversion may be impossible.

- Example: (short) 70000.
- Short values must be between -2^{15} and $2^{15} - 1 = 32767$.

First launch of Ariane 5, 1996

What to do with an impossible conversion?

- Approach 1: Avoid doing it in the first place.
- Approach 2 (Java): Live with a well-defined result.
- Approach 3: Crash.

Example of type conversion put to good use: pseudo-random integers

System method Math.random() returns a pseudo-random double value in [0, 1).

Problem: Given *N*, generate a pseudo-random *integer* between 0 and $N-1$ (inclusive).

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

String to int (system method)

double to int (cast)

int to double (automatic)

```
% java RandomInt 6
3

% java RandomInt 6
0

% java RandomInt 10000
3184
```

Generating Random numbers (pseudo random numbers)

The Math Class: Java API

public static double random()

Returns a double value with a positive sign, **greater than or equal to 0.0 and less than 1.0**: [0, 1)

Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

What is the range of values for each of the following calls.

- 10 * Math.random()

[0.0, 10.0)

- (int)(100 * Math.random() + 1)

[1,100]

Generally speaking: Generate a random integer between a and b inclusive (where b > a)

(int) (Math.random() * (b – a + 1) + a)

Generate a random integer in the following ranges:

- 1 and 20 inclusive

(int)(Math.random() * 20 + 1)

- 50 and 100 inclusive

(int) (Math.random() * 51 + 50)

Summary

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

Commonly-used built-in data types in Java

- String, for computing with *sequence of characters*, for input and output.
- int, for computing with *integers*, for math calculations in programs.
- double, for computing with *floating point numbers*, typically for science and math apps.
- boolean, for computing with *true* and *false*, for decision making in programs.

In Java you must:

- Declare the types of your variables.
- Convert from one type to another when necessary.
- Identify and resolve type errors in order to *compile* your code.

Pay attention to the type of your data.

The Java compiler is your *friend*: it will help you identify and fix type errors in your code.

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational Operators

Relational Operators and operator precedence

LO 3.7

operator	*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

Caution. Do not confuse = with ==

a = b ← Assign the value of b to a. changes a, not b

a == b ← compare the values of a and b.
No change in a or b

Level	Operator	Description	Associativity
16	`[]` `.` `()`	access array element access object member parentheses	left to right
15	`++` `--`	unary post-increment unary post-decrement	not associative
14	`++` `--` `+` `-` `!` `~`	unary pre-increment unary pre-decrement unary plus unary minus unary logical NOT unary bitwise NOT	right to left
13	`()` `new`	cast object creation	right to left
12	`* / %`	multiplicative	left to right
11	`+ -` `+`	additive string concatenation	left to right
10	`<< >>` `>>>`	shift	left to right

9	`< <=`
	`> >=`
	`instanceof`
		relational	not associative

8	`==` `!=`	equality	left to right
7	`&`	bitwise AND	left to right
6	`^`	bitwise XOR	left to right
5	`|`	bitwise OR	left to right
4	`&&`	logical AND	left to right
3	`||`	logical OR	left to right
2	`?:`	ternary	right to left

1	`= += -=`
	`*= /= %=`
	`&= ^= |=`
	`<<= >>= >>>=`
		assignment	right to left

highest

lowest

More Java Practice (iteration preview) – Find the sum of the first 50 counting numbers

```
SET count TO 1
SET sum TO 0
WHILE count < = 50
      ADD count TO sum
     ADD 1 TO count
ENDWHILE
DISPLAY sum
```

More Java Practice – Find the sum of the digits in a number input by user.

Image sources

http://commons.wikimedia.org/wiki/File:KnuthAtOpenContentAlliance.jpg

http://commons.wikimedia.org/wiki/File:Ada_Lovelace.jpg

http://commons.wikimedia.org/wiki/File:Babbages_Analytical_Engine,_1834-1871._(9660574685).jpg

http://commons.wikimedia.org/wiki/File:James_Gosling_2005.jpg

http://commons.wikimedia.org/wiki/File:Bjarne-stroustrup.jpg

http://blog-images.muddymatches.co.uk.s3.amazonaws.com/dating-advice/wp-content/uploads/2013/01/Bad-guy.jpg

Image sources

http://commons.wikimedia.org/wiki/Category:2013_Boston_Red_Sox_season#mediaviewer/File:Koji_Uehara_2_on_June_15,_2013.jpg

http://thenationalforum.org/wp-content/uploads/2011/03/Legendary-Musicians.png

http://pixabay.com/p-15812/?no_redirect

Image sources

http://www.esa.int/spaceinimages/Images/2009/09/Explosion_of_first_Ariane_5_flight_June_4_1996

http://commons.wikimedia.org/wiki/File%3AStethoscope-2.png

3. Programming in Java

- Java Programs
- Built-in data types
- Input and Output
- Boolean and Logical Operators
- The math class
- Casting
- Relational Operators

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

3. Elements of Programming

https://introcs.cs.rutgers.edu