Built-in data types

logical AND	logical OR	logical NOT
&&	||	!

```
public static void main(String [] args)
```
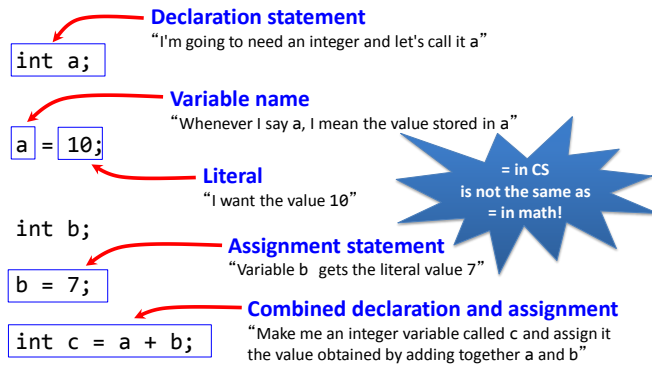

Variables and data types

- Variables
  - Stores information your program needs
  - Each has a unique name
  - Each has a specific type

Java built-in type	what it stores	example values	operations
String	sequence of characters	"Hello world!" "I love this!"	concatenate
char	characters	'a', 'b', '!'	compare
int	integer values	42 1234	add, subtract, multiply, divide, remainder
double	floating-point values	9.95 3.0e8	add, subtract, multiply, divide
boolean	truth values	true false	and, or, not

Some definitions

Declaration statement
"I'm going to need an integer and let's call it a"

```
int a;
```

Variable name
"Whenever I say a, I mean the value stored in a"

```
a = 10;
```

Literal
"I want the value 10"

= in CS
is not the same as
= in math!

```
int b;
```

Assignment statement
"Variable b gets the literal value 7"

```
b = 7;
```

Combined declaration and assignment
"Make me an integer variable called c and assign it the value obtained by adding together a and b"

```
int c = a + b;
```

Text

- String data type
  - A sequence of characters
  - Double quote around the characters
  - Concatenation using the + operator

```
String firstName = "Keith";
String lastName  = "Vertanen";
String fullName  = firstName + " " + lastName;
String favNumber = "42";

System.out.println(fullName +
                "'s favorite number is " +
                favNumber);
```

```
Keith Vertanen's favorite number is 42
```

Characters

- char data type
  - Holds a single character
  - Single apostrophe, e.g. 'a', 'z'

```java
public class CharExample
{
    public static void main(String [] args)
    {
        char ch1 = 'y';
        char ch2 = 'o';
        String result = "" + ch1;
        result += ch2;
        result += ch2;
        result += ch2;
        System.out.println(result);
    }
}
```

```
% java CharExample
yooo
```

Integers

- int data type
  - An integer value between -2^{31} and $+2^{31}-1$
    - Between -2,147,483,648 and 2,147,483,647
  - Operations:

add	subtract	multiply	divide	remainder
+	-	*	/	%

example	result	comment
10 + 7	17	
10 - 7	3	
10 * 7	70	
10 / 7	1	integer division, no fractional part
10 % 7	3	remainder after dividing by 7
10 / 0		runtime error, you can't divide an integer by 0!

Watch out for this! / is integer division if both sides are integers!
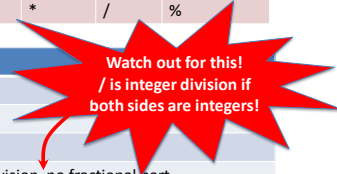
Integers

- int data type
  - Normal rules of mathematical precedence
    - e.g. multiplication/division before addition/subtraction
  - Use ()'s to force order of calculation

example	result	comment
10 + 7 * 2	24	multiplication comes before addition
(10 + 7) * 2	34	()'s force addition to occur first
10 / 7 + 2	3	integer division result is 1 which is added to 2
10 - 7 - 2	1	
(10 - 7) - 2	1	
10 - (7 - 2)	5	

Floating-point numbers

- double data type
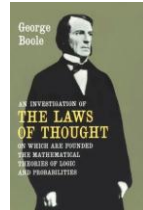 - Floating-point number (as specified by IEEE 754)
  - Operations:

add	subtract	multiply	divide
+	-	*	/

example	result
9.95 + 2.99	12.94
1.0 - 2.0	-1.0
1.0 / 2.0	0.5
1.0 / 3.0	0.3333333333333333
1.0 / 0.0	Infinity
0.0 / 123.45	0.0
0.0 / 0.0	NaN

Booleans

- **boolean data type**
  - Either `true` or `false`
  - Controls logic and flow of control in programs
  - Operations:

George Boole

AN INVESTIGATION OF
THE LAWS
OF THOUGHT
ON WHICH ARE FOUNDED
THE MATHEMATICAL
THEORIES OF LOGIC
AND PROBABILITIES

logical AND	logical OR	logical NOT
&&	||	!

Note: two symbols for logical AND and OR, not one!

Booleans

- **boolean data type**

logical AND	logical OR	logical NOT
&&	||	!

!a   → "Is a set to `false`?"
a && b → "Are both a *and* b set to `true`?"
a || b → "Is either a *or* b (or both) set to `true`?"

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

- Given two numbers → return a `boolean`

operator	meaning	true example	false example
==	equal	7 == 7	7 == 8
!=	not equal	7 != 8	7 != 7
<	less than	7 < 8	8 < 7
<=	less than or equal	7 <= 7	8 <= 7
>	greater than	8 > 7	7 > 8
>=	greater than or equal	8 >= 2	8 >= 10

Is the sum of a, b and c equal to 0?   (a + b + c) == 0
Is grade in the B range?   (grade >= 80.0) && (grade < 90.0)
Is sumItems an even number?   (sumItems % 2) == 0

Leap year example

- Years divisible by 4 but not 100 → leap year
- Years divisible by 400 → leap year

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

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

        // But also leap year if divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);
        System.out.println(isLeapYear);
    }
}
```

% java LeapYear 2000
true

Type conversion

- Java is strongly typed
  - Helps protect you from mistakes (aka "bugs")

```java
public class TypeExample0
{
   public static void main(String [] args)
   {
      int orderTotal;
      double costItem = 29.95;

      orderTotal = costItem * 1.06;
      System.out.println("total=" + orderTotal);
   }
}
```

```
% javac TypeExample0.java
TypeExample0.java:7: possible loss of precision
found   : double
required: int
      orderTotal = costItem * 1.06;
                            ^
```

Type conversion

- Converting from one type to another:
  - Manually → using a cast
    - A cast is accomplished by putting a type inside ()'s
  - Casting to `int` drops fractional part
    - Does not round!

```java
public class TypeExample1
{
   public static void main(String [] args)
   {
      int orderTotal;
      double costItem = 29.95;
      orderTotal = (int) (costItem * 1.06);
      System.out.println("total=" + orderTotal);
   }
}
```

```
% java TypeExample1
total=31
```

Type conversion

- Automatic conversion
  - `String` concatenation using the + operator converts numeric types to also be a `String`
  - Numeric types:
    - If no loss of precision → automatic promotion

```java
public class TypeExample2
{
   public static void main(String [] args)
   {
      double orderTotal;
      int costItem = 30;
      orderTotal = costItem * 1.06;
      System.out.println("total=" + orderTotal);
   }
}
```

```
% java TypeExample2
total=31.8
```

args array

Program input comes in as Strings from command line (for now)

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

```
% java CostCalc bananas 12 0.21
To buy 12 bananas you will need $2.52
```

identifier	meaning	value	type
args[0]	1st thing on command line after Java class name	"bananas"	String
args[1]	2nd thing on command line	"12"	String
args[2]	3rd thing on command line after Java class	"0.21"	String
args.length	# of things on command line	3	int

Static methods

- Java has lots of helper methods
  - Things that take value(s) and return a result
    - e.g. Math functions
    - e.g. Type conversion: `String → int`
      `String → double`
    - e.g. Random number generation
- For now, we'll stick to `static` methods
  - Live in some particular Java class
    - e.g. `Math`, `Integer` or `Double`
  - Call using class name followed by dot

Converting text to a numeric type

method	description
`Integer.parseInt(String a)`	converts text a into an int
`Double.parseDouble(String a)`	convert text a into a double

```java
public class CostCalc
{
    public static void main(String [] args)
    {
        String product = args[0];
        int    qty     = Integer.parseInt(args[1]);
        double cost    = Double.parseDouble(args[2]);

        double total = qty * cost;

        System.out.print("To buy " + qty);
        System.out.print(" " + product);
        System.out.println(" you will need $" + total);
    }
}
```

```
% java CostCalc elections 2 1e6
To buy 2 elections you will need $2000000.0
```

Different types of errors: runtime

runtime error

```
% java CostCalc apples 6 -10
To buy 6 apples you will need $-60.0

% java CostCalc apples 6 foo
Exception in thread "main" java.lang.NumberFormatException: For input
string: "foo"
        at sun.misc.FloatingDecimal.readJavaFormatString
        (FloatingDecimal.java:1222)
        at java.lang.Double.parseDouble(Double.java:510)
        at CostCalc.main(ArgsExample.java:7)

% java CostCalc apples 6.0 0.25
Exception in thread "main" java.lang.NumberFormatException: For input
string: "6.0"
        at java.lang.NumberFormatException.forInputString
        (NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:458)
        at java.lang.Integer.parseInt(Integer.java:499)
        at CostCalc.main(ArgsExample.java:6)
```

Different types of errors: compile time

```java
public class CostCalc
{
    public static void main(String [] args)
    {
        String  product = args[0];
        int     qty     = args[1];
        double  cost    = args[2];
        double  total   = qty * cost;

        System.out.print("To buy " + qty);
        System.out.print(" " + product);
        System.out.println(" you will need $" + total);
    }
}
```

compile time error

```
% javac CostCalc.java
CostCalc.java:6: incompatible types
found   : java.lang.String
required: int
        int     qty     = args[1];
                          ^
CostCalc.java:7: incompatible types
found   : java.lang.String
required: double
        double  cost    = args[2];
                          ^
2 errors
```

Randomness

- Simulate roll of two 6-sided dice
- Generate two random #'s between 1 and 6

`Math.random()`	→ number in [0, 1.0) e.g. 0.0, 0.312, 0.9999999
`Math.random()*6`	→ number in [0, 6.0) e.g. 0.0, 1.872, 5.9999994
`(Math.random()*6)+1`	→ number in [1, 7.0) e.g. 1.0, 2.872, 6.9999994
`(int)(Math.random()*6)+1`	→ number in set {1, 2, 3, 4, 5, 6} e.g. 1, 2, 6

Randomness

- Simulate roll of two 6-sided dice
- Generate two random #'s between 1 and 6

```java
public class TwoDice
{
    public static void main(String [] args)
    {
        int dice1 = (int) (Math.random() * 6) + 1;
        int dice2 = (int) (Math.random() * 6) + 1;
        int sum   = dice1 + dice2;

        System.out.println(dice1 + " + " +
                           dice2 + " = " +
                           sum);
    }
}
```

```
% java TwoDice
3 + 4 = 7
% java TwoDice
1 + 6 = 7
% java TwoDice
1 + 1 = 2
```

Type conversion quiz

- Automatic: no loss of precision
  - int will convert to a `double` if need be
  - `double` cannot automatically convert to `int`
- Manual: cast or using a method

expression	resulting type	resulting value
`(int) 3.14159`		
`Math.round(3.6)`		
`2 * 3.0`		
`2 * (int) 3.0`		
`(int) 2 * 3.0`		

Type conversion quiz

- Automatic: no loss of precision
  - int will convert to a `double` if need be
  - `double` cannot automatically convert to `int`
- Manual: cast or using a method

expression	resulting type	resulting value
`(int) 3.14159`	int	3
`Math.round(3.6)`	long	4
`2 * 3.0`	double	6.0
`2 * (int) 3.0`	int	6
`(int) 2 * 3.0`	double	6.0

String conversion quiz

- `String` conversion, using:
  - Integer.*parseInt*()
  - Double.*parseDouble*()

expression	resulting type	resulting value
Integer.*parseInt*("30")		
Double.*parseDouble*("30")		
Integer.*parseInt*("30.1")		
Double.*parseDouble*("30.1")		
Integer.*parseInt*("$30")		
Double.*parseDouble*(3.14)		

String conversion quiz

- `String` conversion, using:
  - Integer.*parseInt*()
  - Double.*parseDouble*()

expression	resulting type	resulting value
Integer.*parseInt*("30")	int	30
Double.*parseDouble*("30")	double	30.0
Integer.*parseInt*("30.1")	(runtime error, can't parse as int)	
Double.*parseDouble*("30.1")	double	30.1
Integer.*parseInt*("$30")	(runtime error, can't parse as int)	
Double.*parseDouble*(3.14)	(runtime error, 3.14 not a String)	

String concatenation quiz

- + is addition for numeric types
- + is concatenation for `String` type
- numeric types convert to `String` if needed
  - Strings never (automatically) go back to number

expression	resulting type	resulting value
"testing " + 1 + 2 + 3		
"3.1" + 4159		
"2" + " + " + "3"		
1 + 2 + 3 + "66"		

String concatenation quiz

- + is addition for numeric types
- + is concatenation for `String` type
- numeric types convert to `String` if needed
  - Strings never (automatically) go back to number

expression	resulting type	resulting value
"testing " + 1 + 2 + 3	String	"testing 123"
"3.1" + 4159	String	"3.14159"
"2" + " + " + "3"	String	"2 + 3"
1 + 2 + 3 + "66"	String	"666"

Summary

- Variables
  - Allows us to store and compute on data
  - `String, int, double, boolean, char`
  - Boolean operators for logic and program flow control (more on this next time!)
- Type conversion
  - Automatic
    - e.g. `int` converting itself to a `double`
  - Explicit via cast or method call
  - Important:
    - Cause of many, many, many software bugs