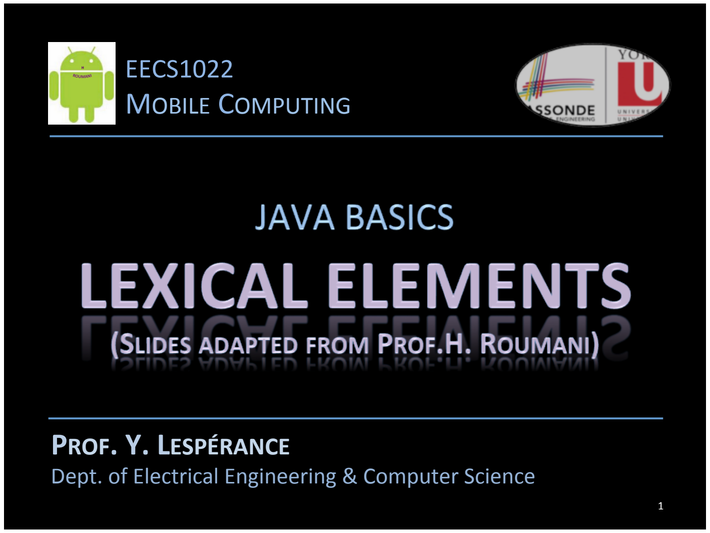


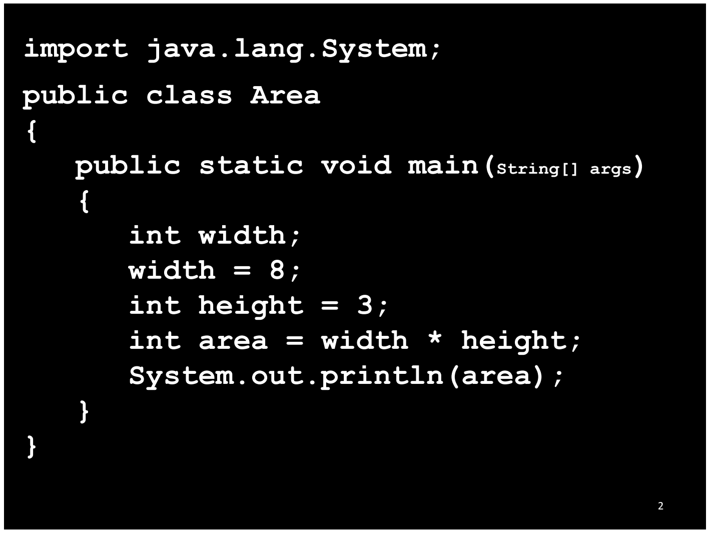
EECS1022
MOBILE COMPUTING

JAVA BASICS

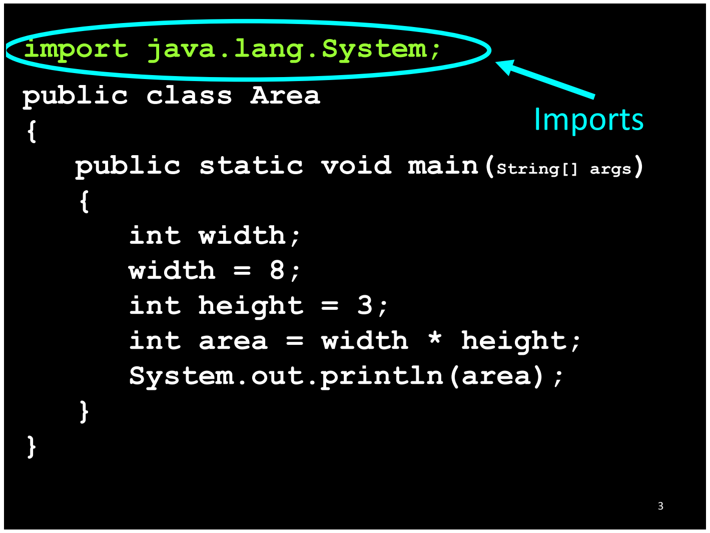
LEXICAL ELEMENTS

(SLIDES ADAPTED FROM PROF.H. ROUMANI)

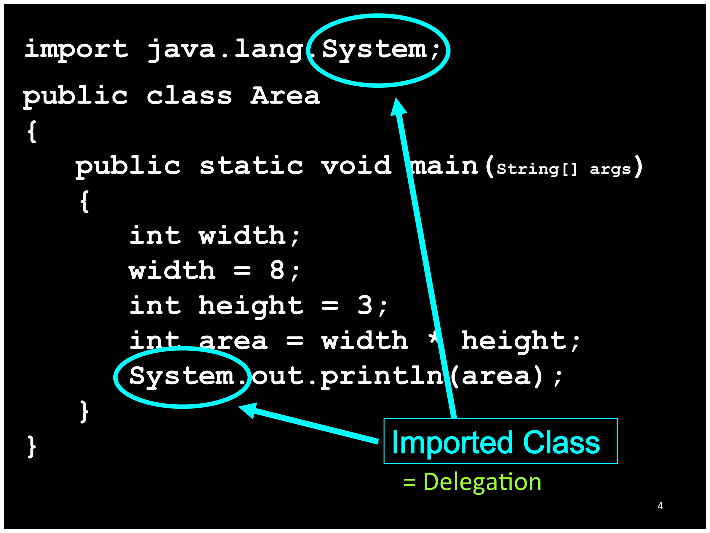
PROF. Y. LESPÉRANCE
Dept. of Electrical Engineering & Computer Science



```
import java.lang.System;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}
```

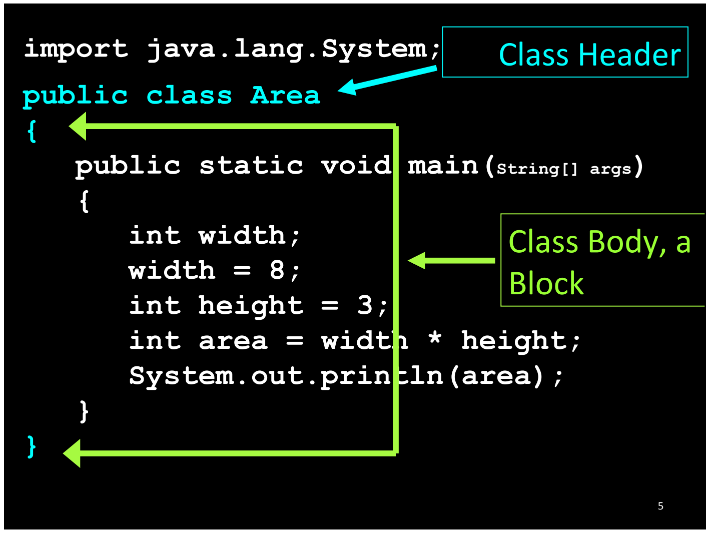


```
import java.lang.System;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}
```

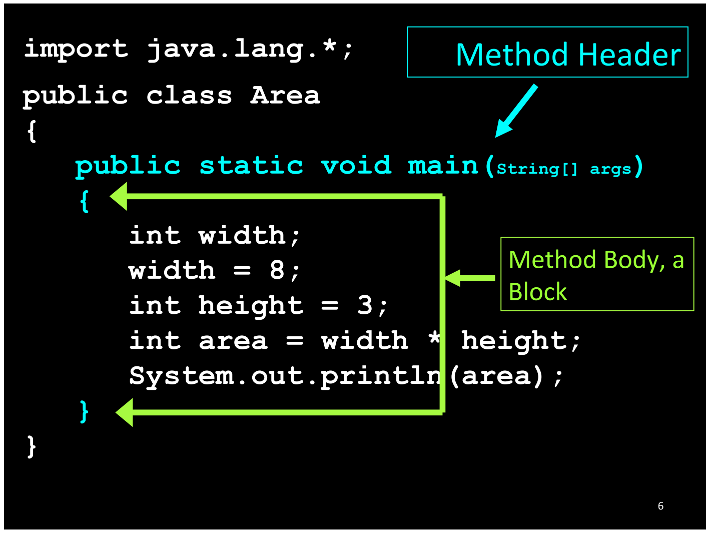


```
import java.lang.System;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}
```

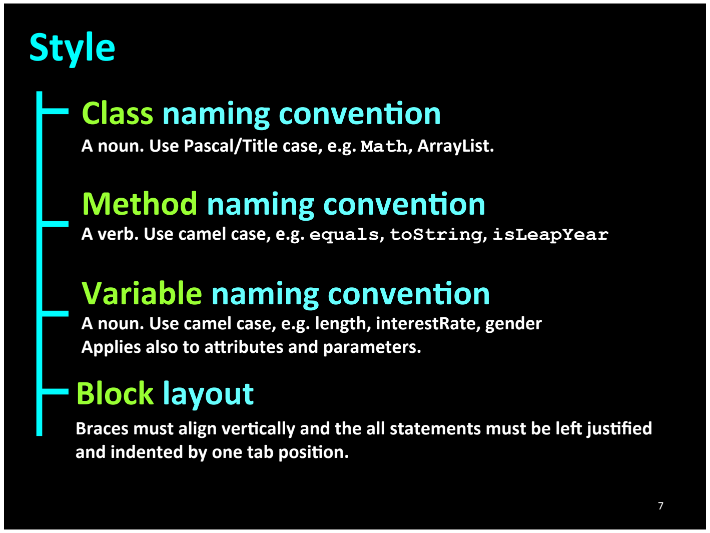
Imported Class
= Delegation



```
import java.lang.System;
public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

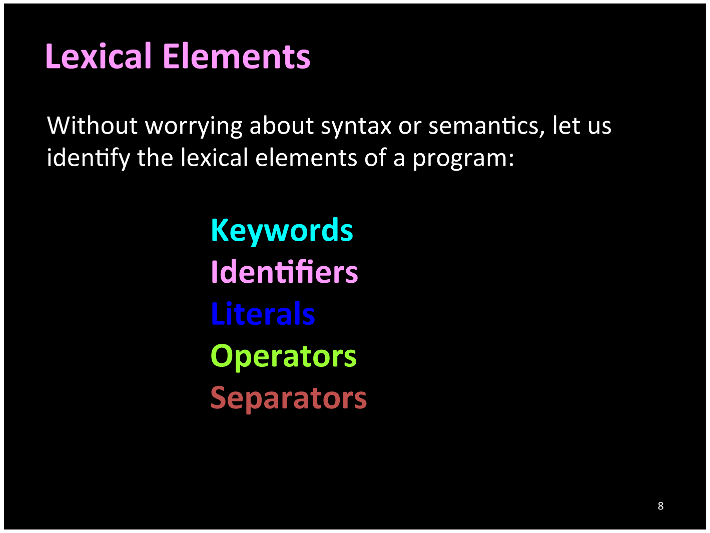


```
import java.lang.*;
public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```



Style

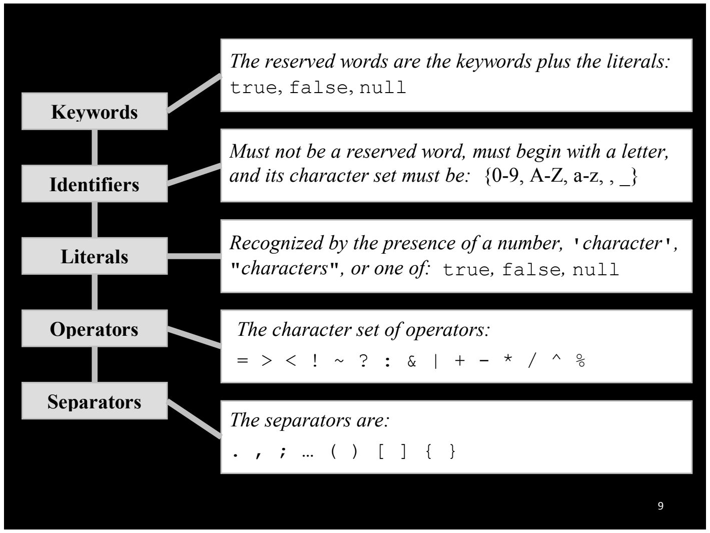
- **Class naming convention**
  A noun. Use Pascal/Title case, e.g. `Math`, ArrayList.
- **Method naming convention**
  A verb. Use camel case, e.g. `equals`, `toString`, `isLeapYear`
- **Variable naming convention**
  A noun. Use camel case, e.g. length, interestRate, gender
  Applies also to attributes and parameters.
- **Block layout**
  Braces must align vertically and the all statements must be left justified and indented by one tab position.



Lexical Elements

Without worrying about syntax or semantics, let us identify the lexical elements of a program:

Keywords
Identifiers
Literals
Operators
Separators



Keywords

Identifiers

Literals

Operators

Separators

The reserved words are the keywords plus the literals: `true`, `false`, `null`

Must not be a reserved word, must begin with a letter, and its character set must be: {0-9, A-Z, a-z, , _}

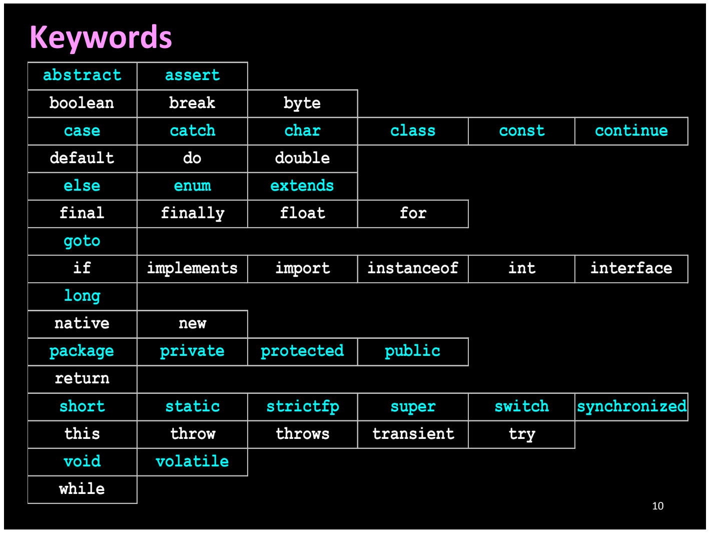
Recognized by the presence of a number, `'`*character*`'`, `"`*characters*`"`, *or one of:* `true`, `false`, `null`

The character set of operators:

`= > < ! ~ ? : & | + - * / ^ %`

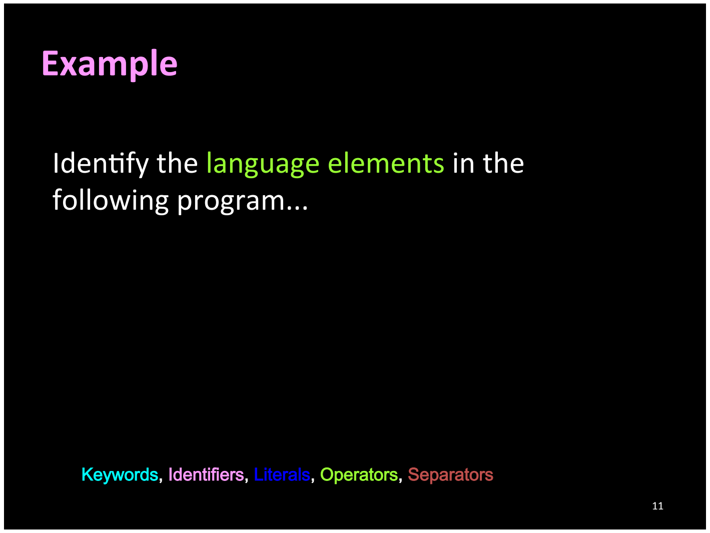
The separators are:

`. , ; … ( ) [ ] { }`



Keywords

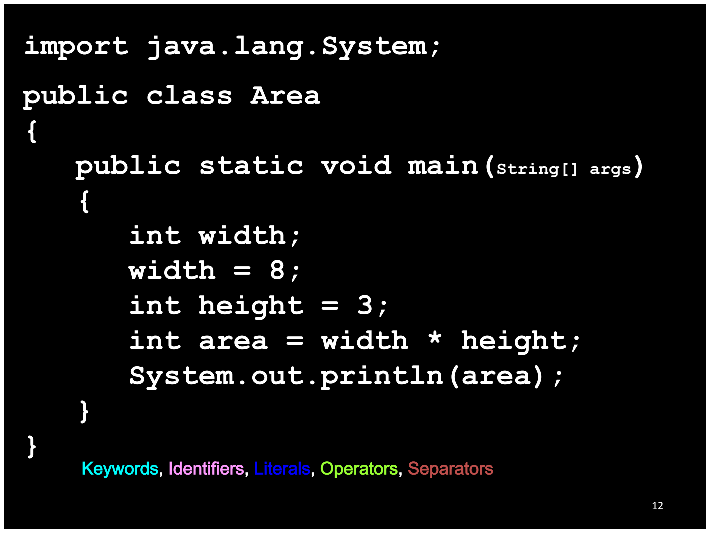
abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					



Example

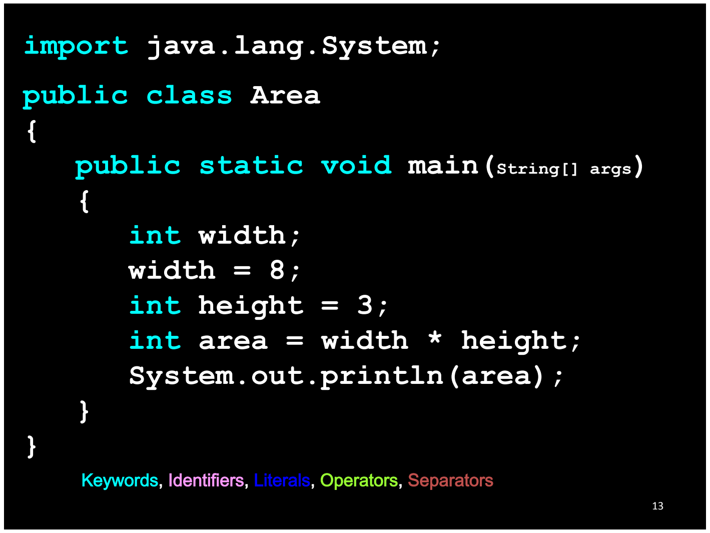
Identify the language elements in the following program...

Keywords, Identifiers, Literals, Operators, Separators



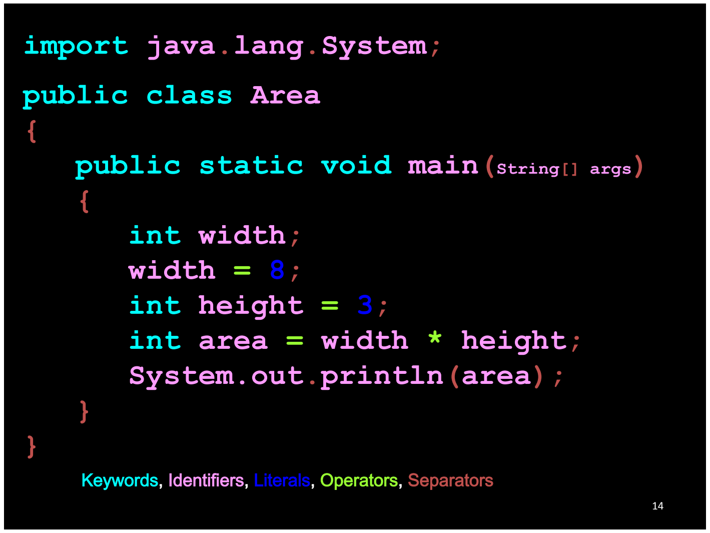
```
import java.lang.System;
public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

Keywords, Identifiers, Literals, Operators, Separators



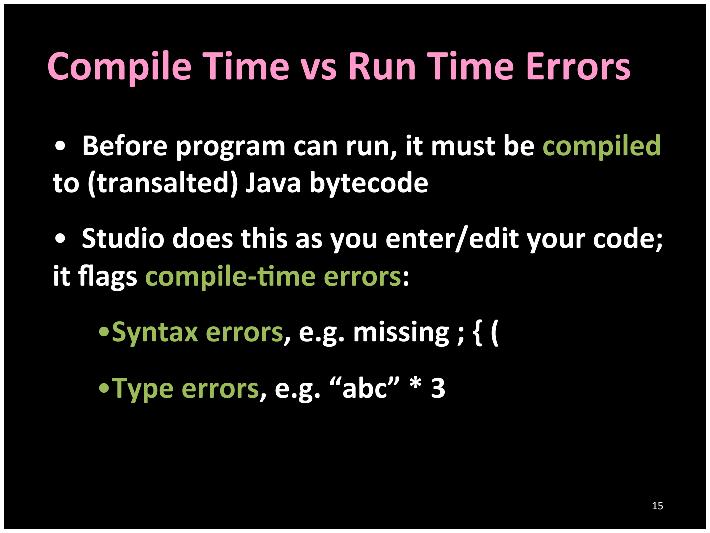
```
import java.lang.System;
public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

Keywords, Identifiers, Literals, Operators, Separators



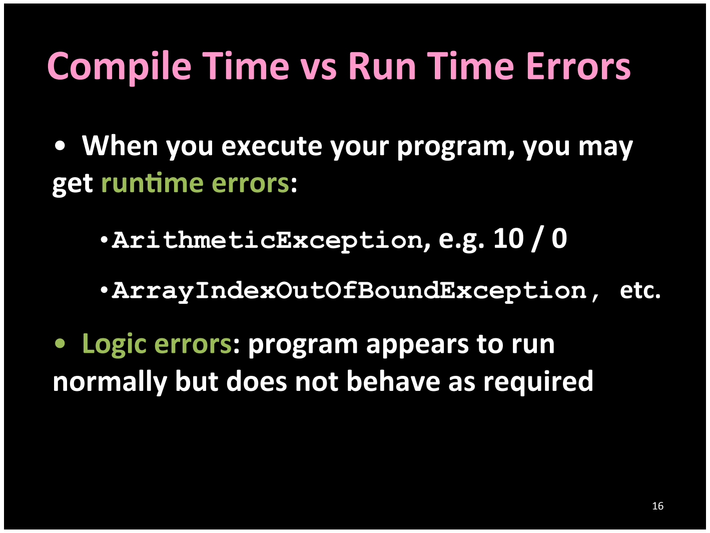
```
import java.lang.System;
public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

Keywords, Identifiers, Literals, Operators, Separators



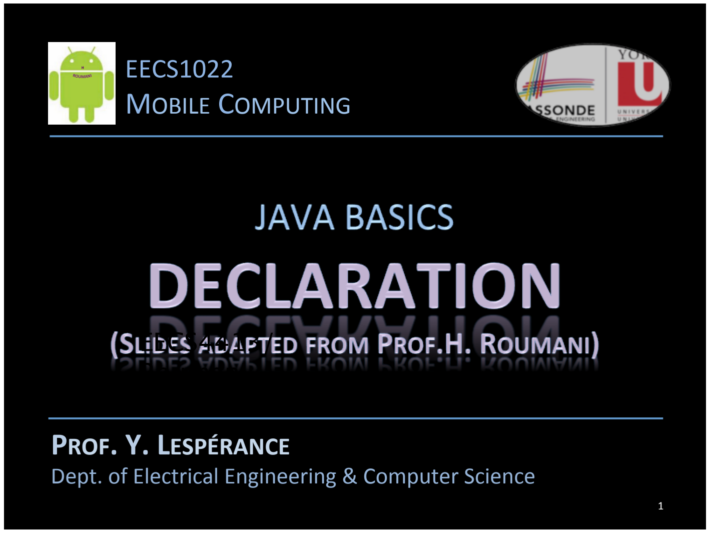
Compile Time vs Run Time Errors

- Before program can run, it must be compiled to (transalted) Java bytecode
- Studio does this as you enter/edit your code; it flags compile-time errors:
  - Syntax errors, e.g. missing ; { (
  - Type errors, e.g. “abc” * 3



Compile Time vs Run Time Errors

- When you execute your program, you may get runtime errors:
  - `ArithmeticException`, e.g. 10 / 0
  - `ArrayIndexOutOfBoundException`, etc.
- Logic errors: program appears to run normally but does not behave as required



EECS1022
Mobile Computing

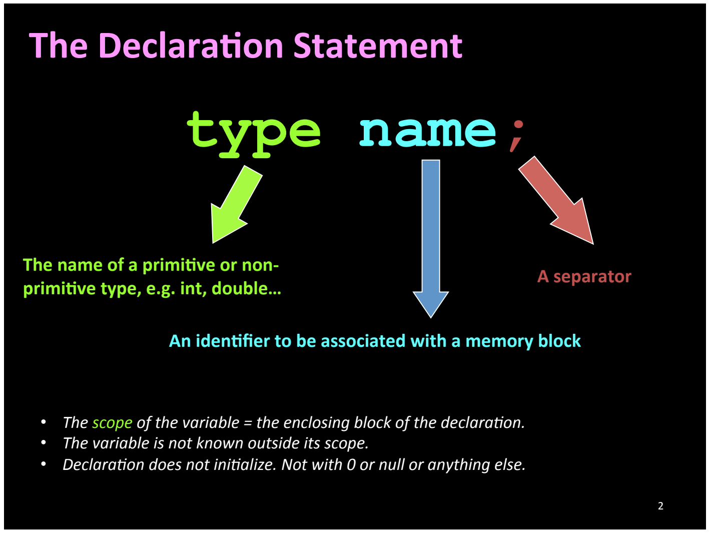
JAVA BASICS

DECLARATION

(Slides adapted from Prof.H. Roumani)

Prof. Y. Lespérance
Dept. of Electrical Engineering & Computer Science

1



The Declaration Statement

```
type name;
```

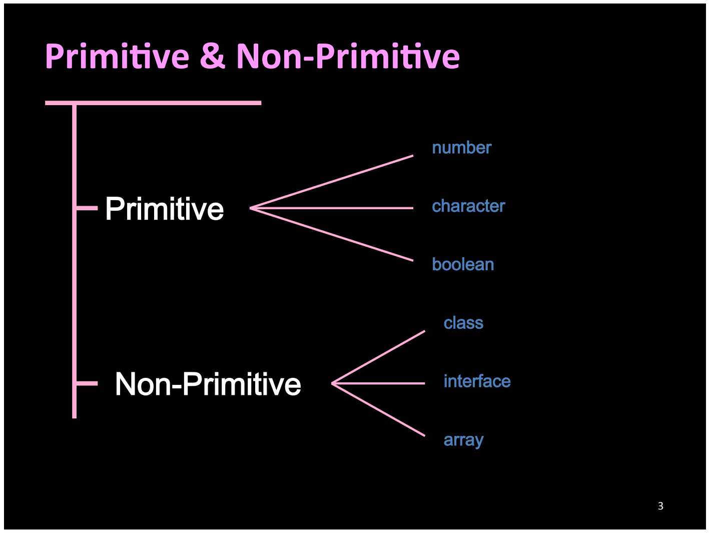
The name of a primitive or non-primitive type, e.g. int, double…

An identifier to be associated with a memory block

A separator

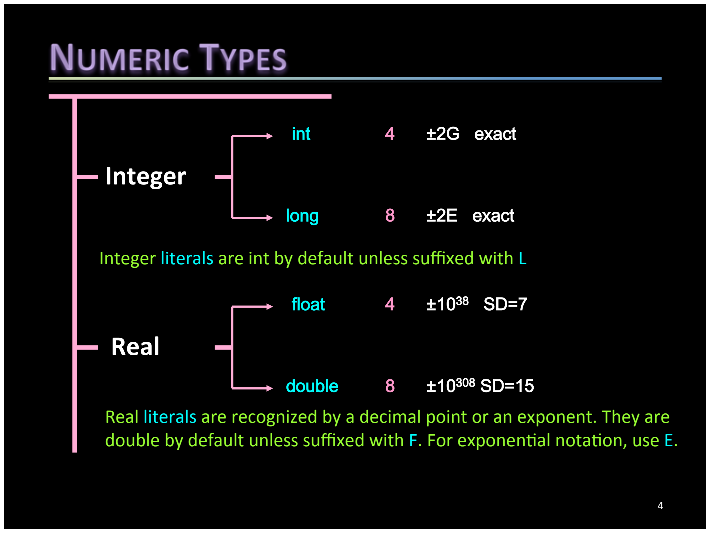
- *The scope of the variable = the enclosing block of the declaration.*
- *The variable is not known outside its scope.*
- *Declaration does not initialize. Not with 0 or null or anything else.*

2



Primitive & Non-Primitive

3



Numeric Types

Integer

int 4 ±2G exact

long 8 ±2E exact

Integer literals are int by default unless suffixed with L

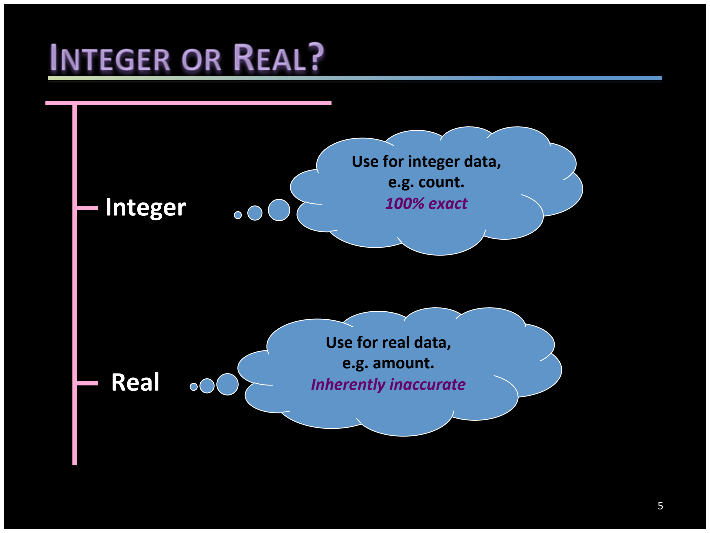
Real

float 4 $\pm10^{38}$ SD=7

double 8 $\pm10^{308}$ SD=15

Real literals are recognized by a decimal point or an exponent. They are double by default unless suffixed with F. For exponential notation, use E.

4



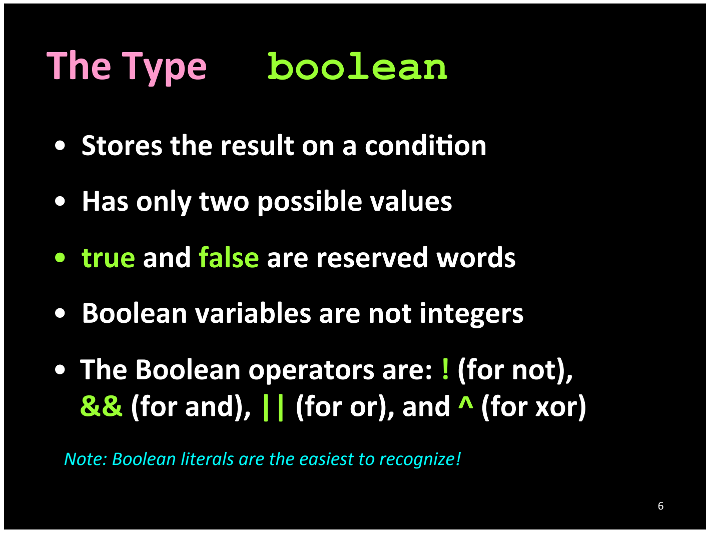
Integer or Real?

Integer

Use for integer data, e.g. count. *100% exact*

Real

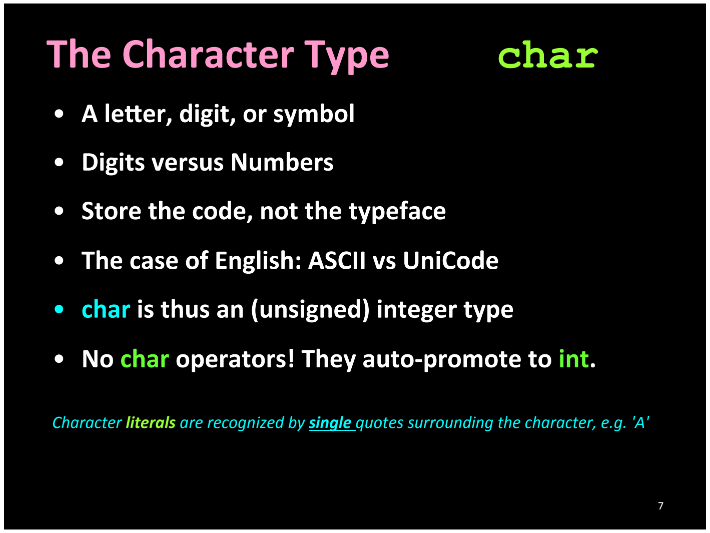
Use for real data, e.g. amount. *Inherently inaccurate*



The Type `boolean`

- Stores the result on a condition
- Has only two possible values
- true and false are reserved words
- Boolean variables are not integers
- The Boolean operators are: ! (for not), && (for and), || (for or), and ^ (for xor)

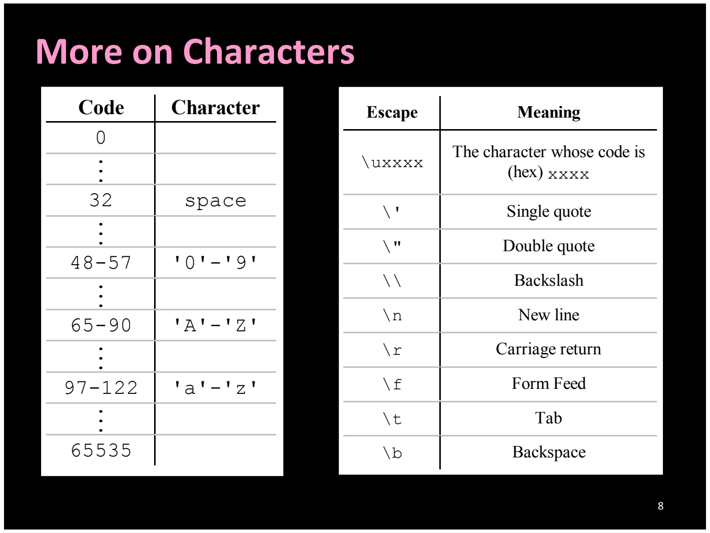
Note: Boolean literals are the easiest to recognize!



The Character Type `char`

- A letter, digit, or symbol
- Digits versus Numbers
- Store the code, not the typeface
- The case of English: ASCII vs UniCode
- char is thus an (unsigned) integer type
- No char operators! They auto-promote to int.

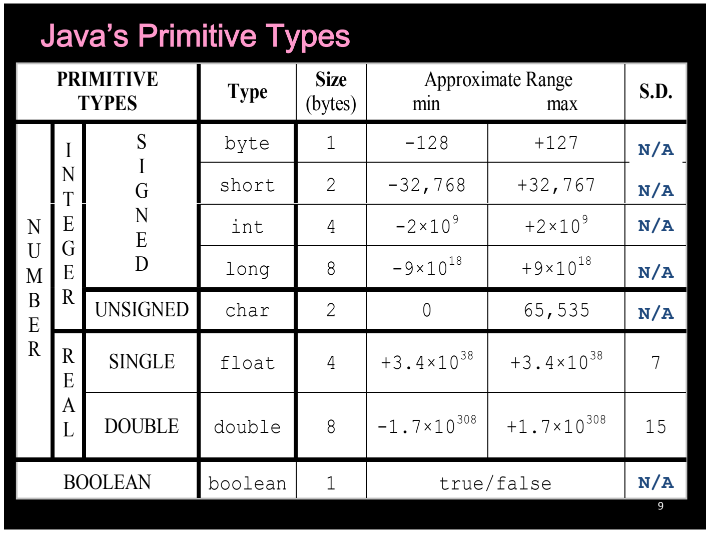
*Character **literals** are recognized by **single** quotes surrounding the character, e.g. 'A'*



More on Characters

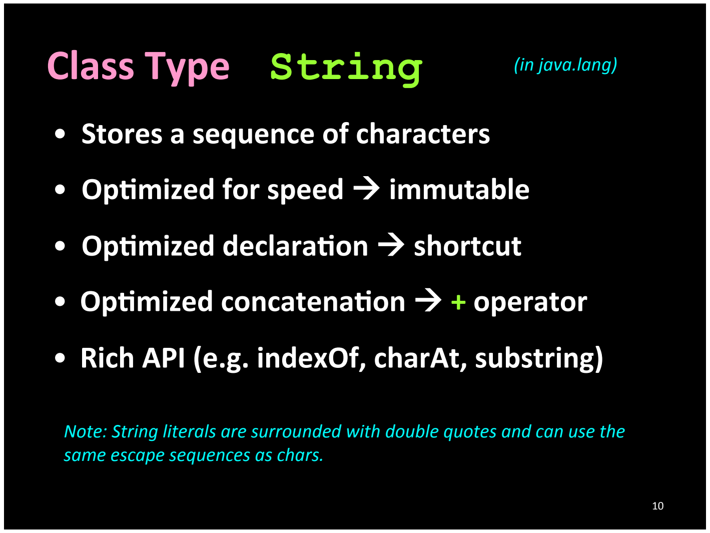
Code	Character
0	
⋮	
32	space
⋮	
48-57	'0'-'9'
⋮	
65-90	'A'-'Z'
⋮	
97-122	'a'-'z'
⋮	
65535	

Escape	Meaning
\uxxxx	The character whose code is (hex) xxxx
\'	Single quote
\"	Double quote
\\	Backslash
\n	New line
\r	Carriage return
\f	Form Feed
\t	Tab
\b	Backspace



Java’s Primitive Types

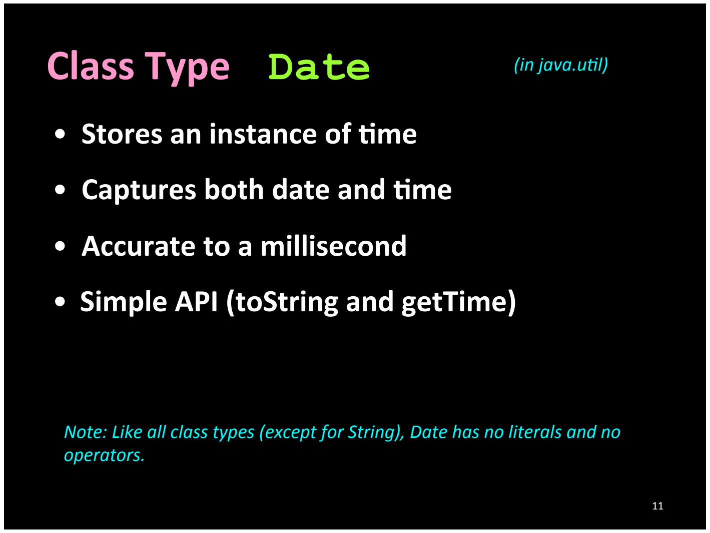
PRIMITIVE TYPES			Type	Size (bytes)	Approximate Range min	max	S.D.
NUMBER	INTEGER	SIGNED	byte	1	-128	+127	N/A
			short	2	-32,768	+32,767	N/A
			int	4	-2×10^{9}	$+2\times10^{9}$	N/A
			long	8	-9×10^{18}	$+9\times10^{18}$	N/A
		UNSIGNED	char	2	0	65,535	N/A
	REAL	SINGLE	float	4	$+3.4\times10^{38}$	$+3.4\times10^{38}$	7
		DOUBLE	double	8	-1.7×10^{308}	$+1.7\times10^{308}$	15
BOOLEAN			boolean	1	true/false		N/A



Class Type `String` *(in java.lang)*

- Stores a sequence of characters
- Optimized for speed → immutable
- Optimized declaration → shortcut
- Optimized concatenation → + operator
- Rich API (e.g. indexOf, charAt, substring)

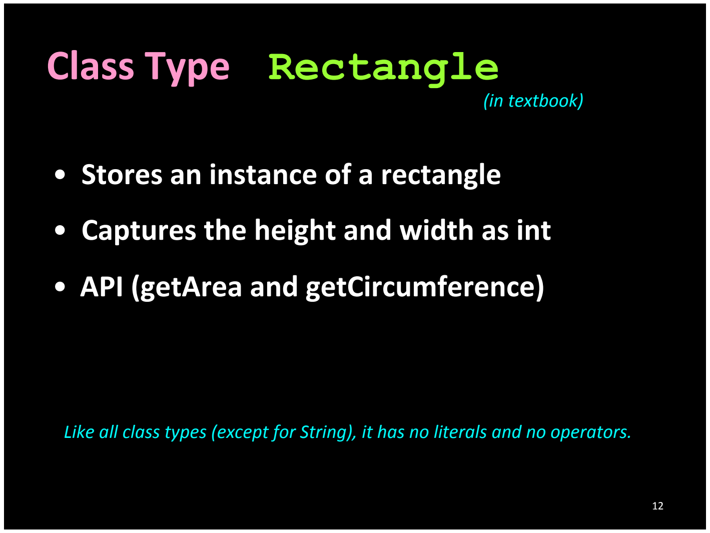
Note: String literals are surrounded with double quotes and can use the same escape sequences as chars.



Class Type `Date` *(in java.util)*

- Stores an instance of time
- Captures both date and time
- Accurate to a millisecond
- Simple API (toString and getTime)

Note: Like all class types (except for String), Date has no literals and no operators.

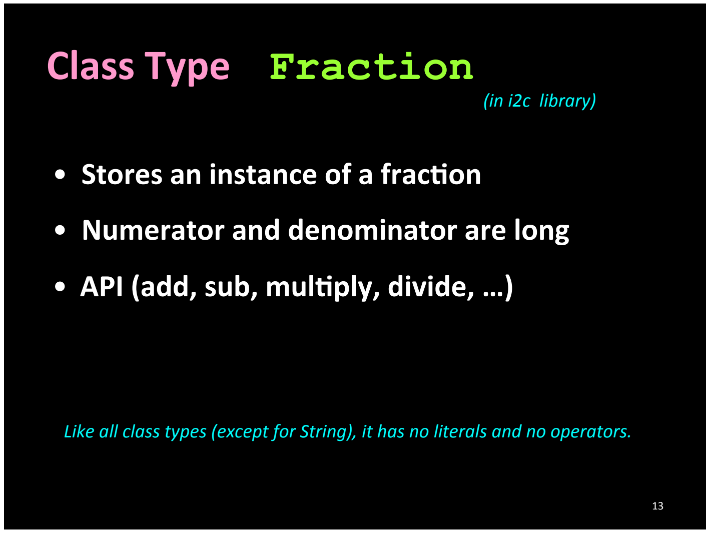


Class Type `Rectangle`

(in textbook)

- Stores an instance of a rectangle
- Captures the height and width as int
- API (getArea and getCircumference)

Like all class types (except for String), it has no literals and no operators.

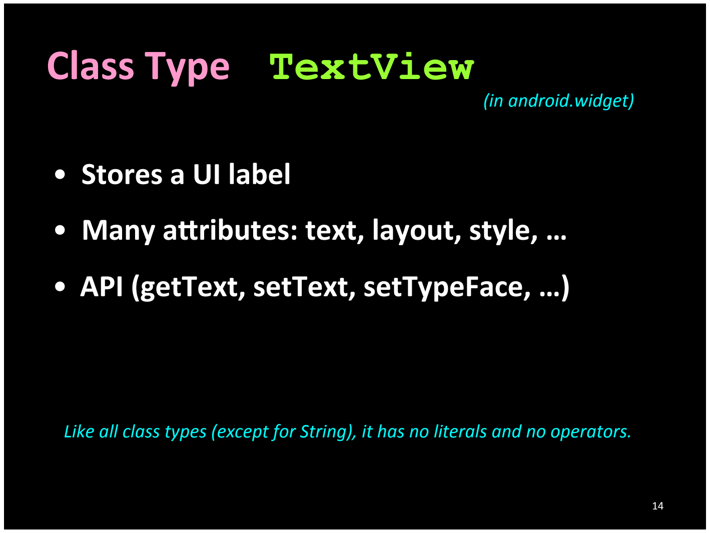


Class Type `Fraction`

(in i2c library)

- Stores an instance of a fraction
- Numerator and denominator are long
- API (add, sub, multiply, divide, …)

Like all class types (except for String), it has no literals and no operators.

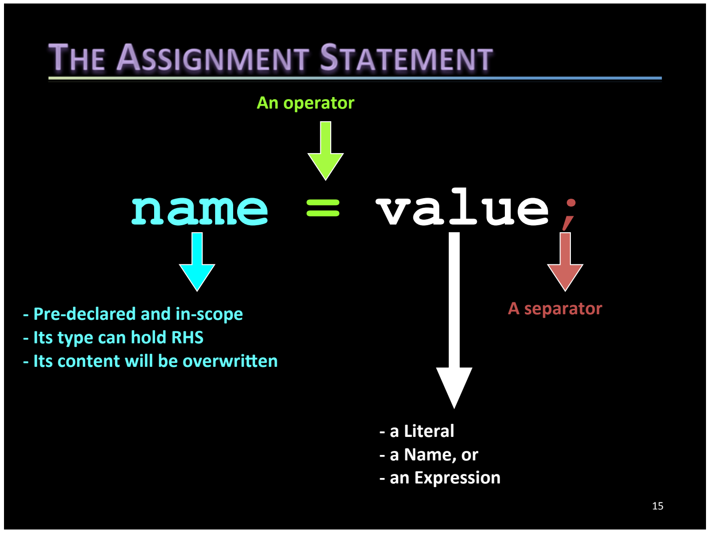


Class Type `TextView`

(in android.widget)

- Stores a UI label
- Many attributes: text, layout, style, …
- API (getText, setText, setTypeFace, …)

Like all class types (except for String), it has no literals and no operators.



The Assignment Statement

An operator

```
name = value;
```

- Pre-declared and in-scope
- Its type can hold RHS
- Its content will be overwritten

A separator

- a Literal
- a Name, or
- an Expression