

# Building Blocks

**Variables**

# Variables (terminology)

- **variable** is a name for a piece of memory which stores data
- to **declare a variable** means to state variable type and give it a name:

```
int x;
```

- to **initialize a variable** means to give a variable a value:

```
int x = 5;
```

- name of the variable (method, class, interface, package...) is called **identifier**

# Identifier rules

1. Must begin with a letter, currency symbol (\$, €, £), or underscore (\_)
2. Can include numbers, but not start with a number
3. Single underscore (\_) is not allowed as an identifier
4. You cannot use the reserved word (see next slide)

\$myVAR12    \_€name\_    \_\_someName1\$\_    BIG\_NAME    \_12x

# Reserved words

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

# Naming conventions

- for variables, use camelCase
- for constants, use SNAKE\_CASE
- identifiers of classes, interfaces, enums records start with first uppercase letter
  - e.g. MyClass, MyInterface, StudentRecord
- identifiers variables and methods start with first lowercase letter
  - e.g. fullName, getFullName()
- Java identifiers are case sensitive !!

```
// multiple variables can be declared and/or initialized
```

```
// in a single line (bad practice, but it compiles)
```

```
int x, y;
```

```
String firstName = "John", lastName = "Wayne";
```

```
boolean v = true, w, z = false;
```

```
// you cannot declare variables of different type in a single line
```

```
int x, String name; // DOES NOT COMPILE
```

```
// "single line" means within the same command, e.g.
```

```
int x,
```

```
    String name; // "same line", DOES NOT COMPILE
```

# Three kinds of variables

1. **local variables** - exist only within the block of code `{ . . . }`
  2. **instance variables (fields)** - defined within the specific instance of the object
  3. **class variables** - belong to a class and is shared with all instances of the class
    - marked with keyword `static`
- instance and class variables don't require initialization
    - assume the default values of their type

// local variables must be initialized before use !!

```
public int doesNotCompile() {
```

```
    int a = 5;
```

```
    int b;
```

```
    return a + b;
```

```
}
```

you are trying to use  
uninitialized variable b



```
public int doesCompile() {
```

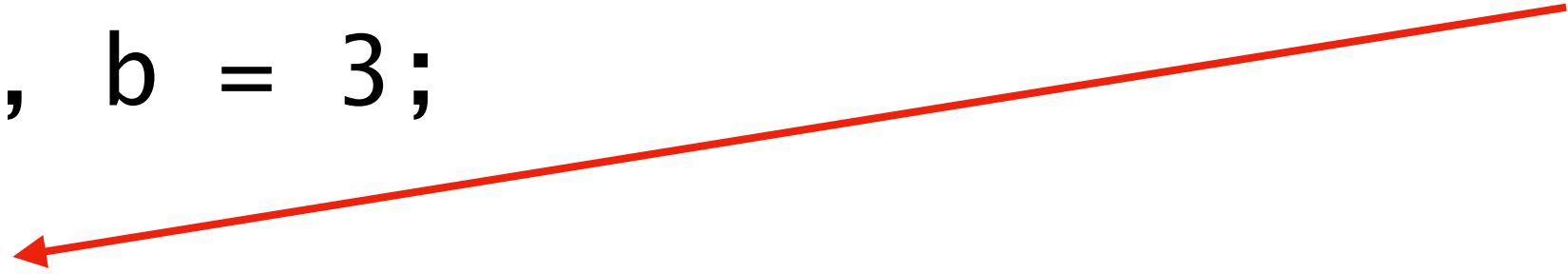
```
    int a = 5, b = 3;
```

```
    int c;
```

```
    return a + b;
```

```
}
```

c is not initialized,  
but it's never used, so this code compiles





// be careful if initialization is within if-statement

```
public void doesNotCompile (boolean isOK) {
```

```
    int a;
```

```
    if (isOK) a = 5;
```

```
    // some code using a
```

```
}
```

might never be reached



```
public void doesCompile (boolean isOk) {
```

```
    int a;
```

```
    if (isOk) a = 5;
```

```
    else a = 2;
```

```
    // some code using a
```

```
}
```

```
// final variables (constants)
```

```
final int MAX_HEIGHT = 100;
```

```
// final can be applied to a reference:
```

```
final int[] MY_NUMBERS = new int[5];
```

```
// reference cannot be modified, but the content of the object can:
```

```
MY_NUMBERS[2] = 13;      // OK
```

```
MY_NUMBERS = null;      // DOES NOT COMPILE
```

# Variable scope

- variables can go out of scope ("cease to exist")
1. *Local variables:* in scope from { to }
  2. *Method parameters:* in scope for the duration of the method
  3. *Instance variables:* in scope from declaration until the object is eligible for garbage collector
  4. *Class variables:* in scope from declaration until the program ends

// simple example:

```
if (isOk) {  
    int x = 5;  
  
    System.out.println("x = " + x); // OK  
}  
  
System.out.println("This is x: " + x);  
  
// DOES NOT COMPILE (x is out of scope)
```