Java Overview

An introduction to the Java Programming Language

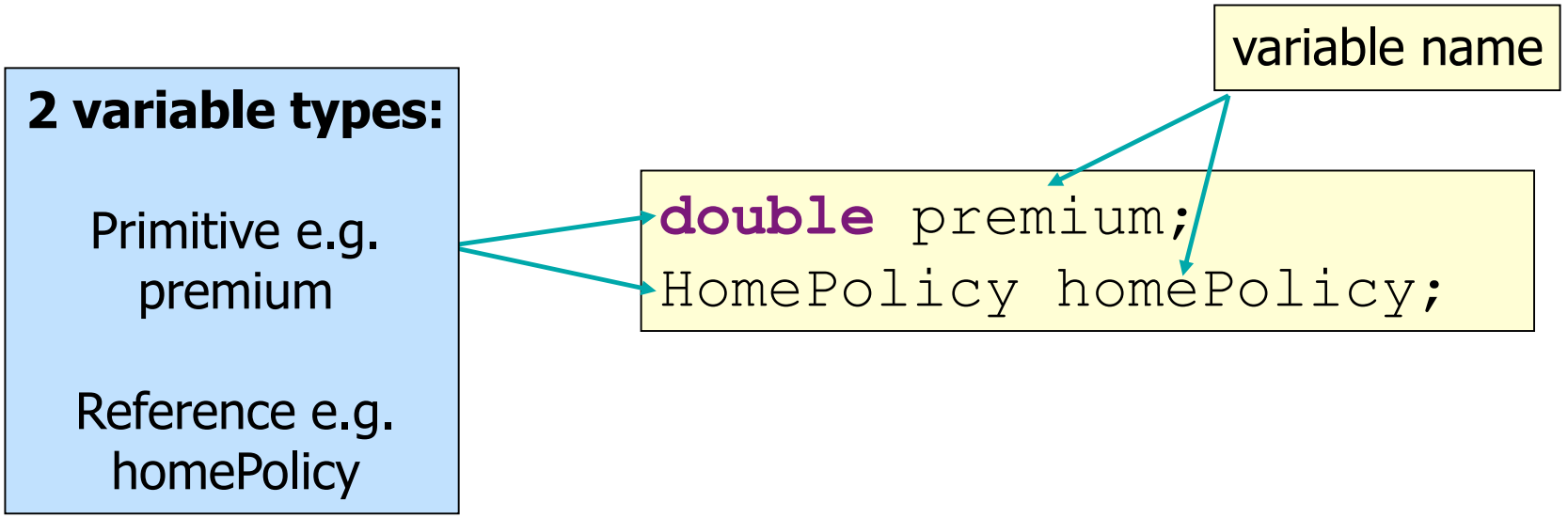
Produced by: Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhan Drohan (sdrohan@wit.ie)

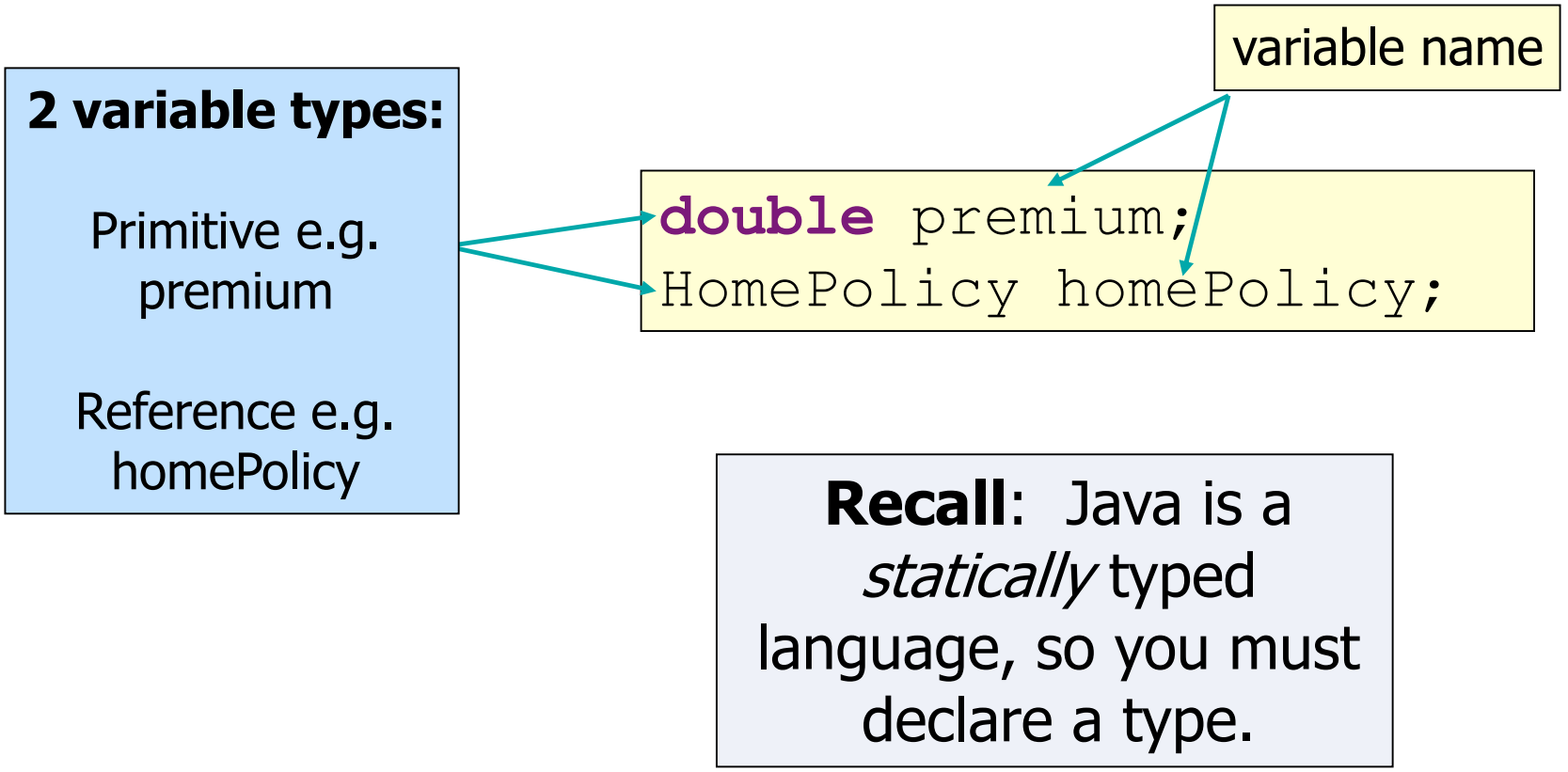
Overview: Road Map

- Java Introduction
  - History
  - Portability
  - Compiler
  - Java Virtual Machine
  - Garbage collection
- Java Syntax
  - Identifiers
  - Expressions
  - Comments
- Java Basics
  - Java types
  - Primitives
  - Objects
  - Variables
  - Operators
  - Identity and equality
- Arrays
  - What are arrays?
  - Creating arrays
  - Using arrays

Java and Types

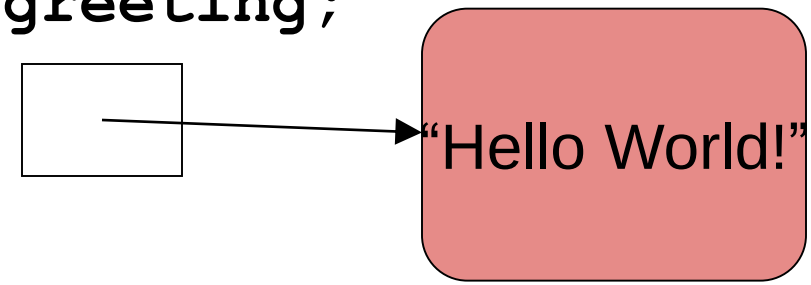


Java and Types



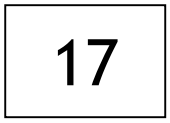
Primitive and Reference Type

```
String greeting;
```



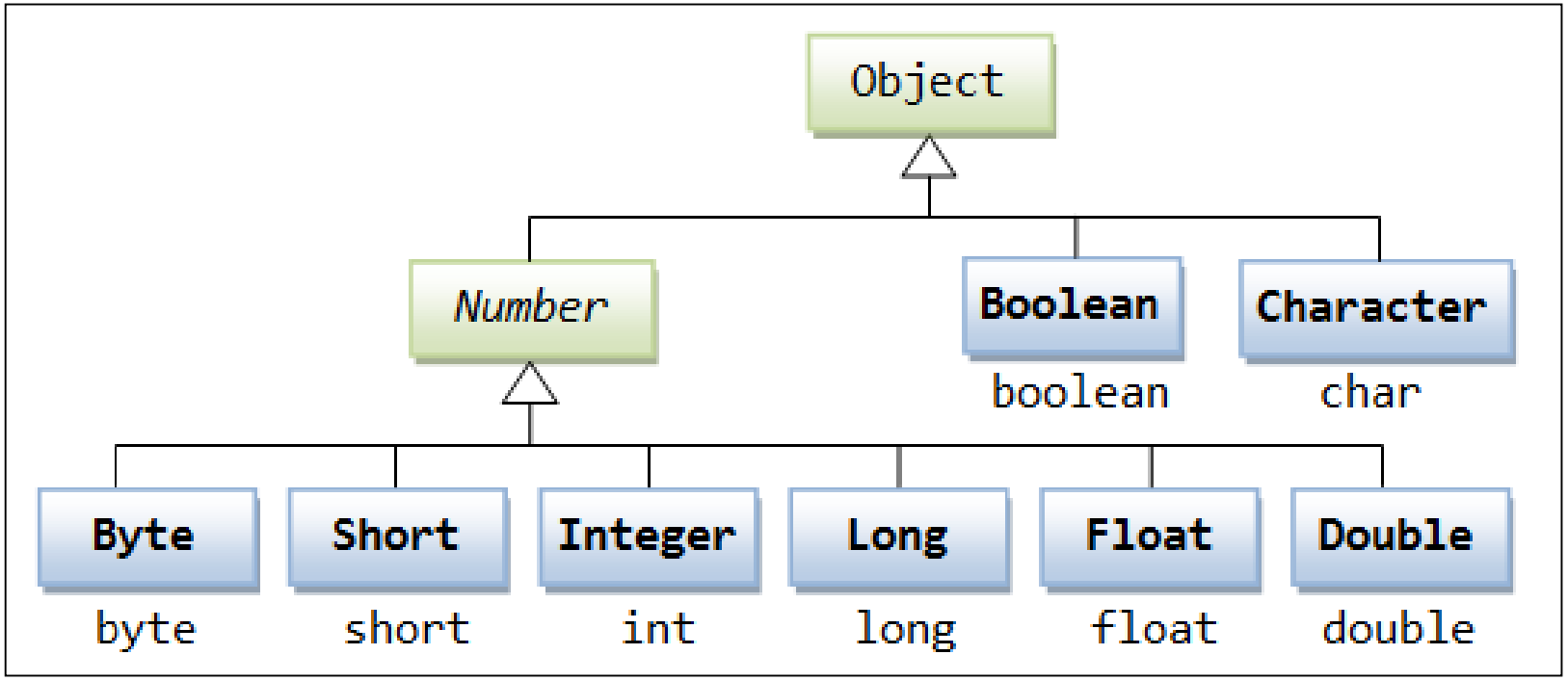
String is a reference type. The **greeting** variable contains a reference to where the String is stored in memory.

```
double premium;
```



premium is a primitive type

Primitives and Wrappers



Wrappers are located in **java.lang** package

Primitive Types

Keyword	Size	Min value	Max value
boolean	true/false		
byte	8-bit	-128	127
short	16-bit	-32768	32767
char	16-bit Unicode		
int	32-bit	-2147483648	2147483647
float	32-bit		
double	64-bit		
long	64-bit	- 9223372036854775808	9223372036854775807

Primitive Operators

Keyword	Description	Keyword	Description	Keyword	Description
+	add	**<**	lesser	**&**	and
-	subtract	**>**	greater	**|**	or
*****	multiple	**=**	assignment	**^**	xor
/	divide	**>=**	greater equal	**!**	not
%	reminder	**<=**	less equal	**&&**	lazy and
(…)	the code within is executed first	**==**	equals	**||**	lazy or
++op	increment first	**!=**	not equal	**<<**	left bit shift
--op	decrement first	**x+=2**	**x=x+2**	**>>**	right bit shift
op++	increment after	**x-=2**	**x=x-2**	**>>>**	right bit shift with zeros
op--	decrement after	**x*=2**	**x=x*2**		

Primitive Operators

Keyword	Description	Keyword	Description	Keyword	Description
+	add	**<**	lesser	**&**	and
-	subtract	**>**	greater	**|**	or
*****	multiple	**=**	assignment	**^**	xor
/	divide	**>=**	greater equal	**!**	not
%	reminder	**<=**	less equal	**&&**	lazy and
(…)	the code within is executed first	**==**	equals	**||**	lazy or
++op	increment first	**!=**	not equal	**<<**	left bit shift
--op	decrement first	**x+=2**	**x=x+2**	**>>**	right bit shift
op++	increment after	**x-=2**	**x=x-2**	**>>>**	right bit shift with zeros
op--	decrement after	**x*=2**	**x=x*2**		

Recall: Java is a *strongly* typed language, so the type will dictate which operators are available to it.

boolean

boolean used in control statements

true or false.

boolean and logical operators

boolean used in control statements

true or false.

Keyword	Description
!	complement
&	and
|	or
^	exclusive or
&&	lazy and
||	lazy or

```
!true              //false
true & true        //true
true | false       //true
```

```
false ^ true       //true
true ^ false       //true
false ^ false      //false
true ^ true        //false
```

XOR: outputs true only when inputs differ (one is true, the other is false)

```
false && true  //false, second operand does not evaluate
true || false  //true, second operand does not evaluate
```

char Type

- Character literals appear in single quotes, and include:
  - Typed characters, e.g. `'z'`
  - Unicode, e.g. `'\u0040'`, equal to `'@'`
  - Escape sequence, e.g. `'\n'`

U+0044	D	01000100	LATIN CAPITAL LETTER D
U+0045	E	01000101	LATIN CAPITAL LETTER E
U+0046	F	01000110	LATIN CAPITAL LETTER F
U+0047	G	01000111	LATIN CAPITAL LETTER G
U+0048	H	01001000	LATIN CAPITAL LETTER H
U+0049	I	01001001	LATIN CAPITAL LETTER I
U+004A	J	01001010	LATIN CAPITAL LETTER J
U+004B	K	01001011	LATIN CAPITAL LETTER K
U+004C	L	01001100	LATIN CAPITAL LETTER L

Escape Sequence Characters

Escape sequence	Unicode	Description
\n	\u000A	New line
\t	\u0009	Tab
\b	\u0008	Backspace
\r	\u000D	Return
\f	\u000C	Form feed
****	\u005C	Backslash
\’	\u0027	Single quote
\”	\u0022	Double quote

Escape Sequence Characters

Escape sequence	Unicode	Description
\n	\u000A	New line
\t	\u0009	Tab
\b	\u0008	Backspace
\r	\u000D	Return
\f	\u000C	Form feed
****	\u005C	Backslash
\'	\u0027	Single quote
\"	\u0022	Double quote

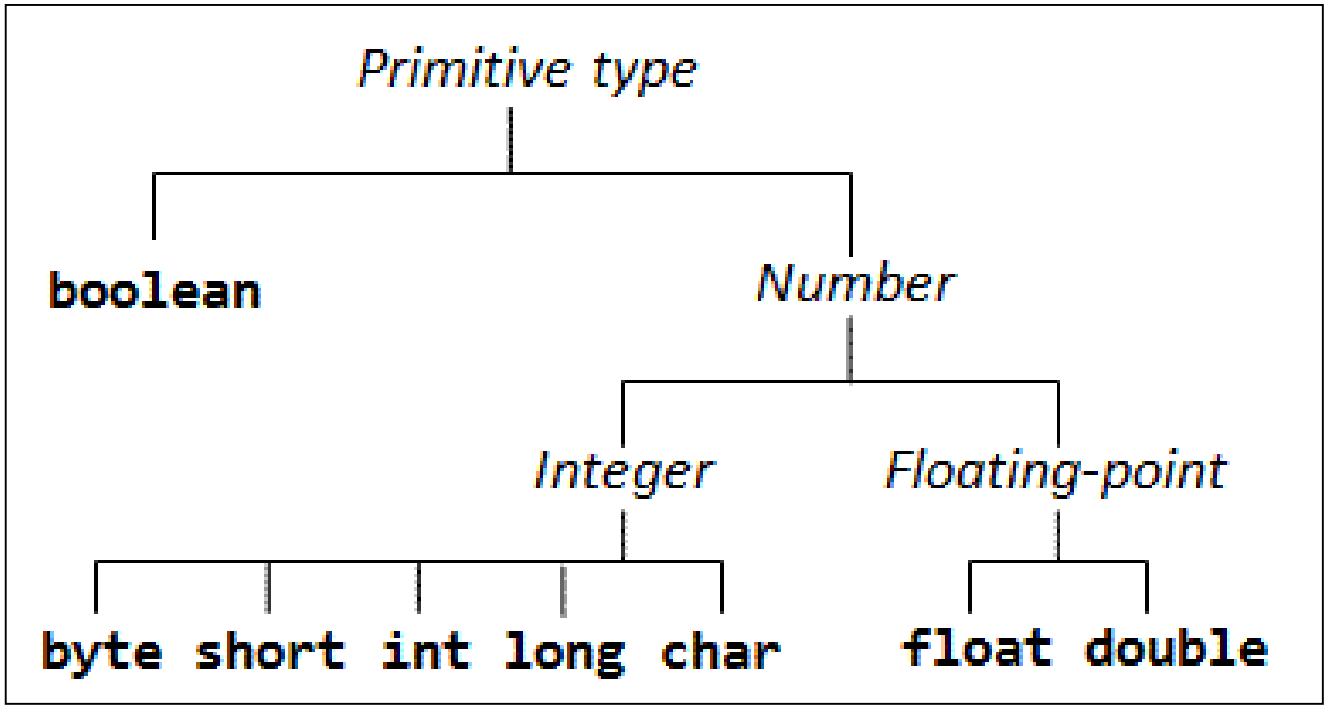
```
System.out.println("Hello");
System.out.println("World");
```

```
System.out.println("Hello\nWorld");
```

Same output:

Hello
World

Numeric Types



Manipulating Numeric Types

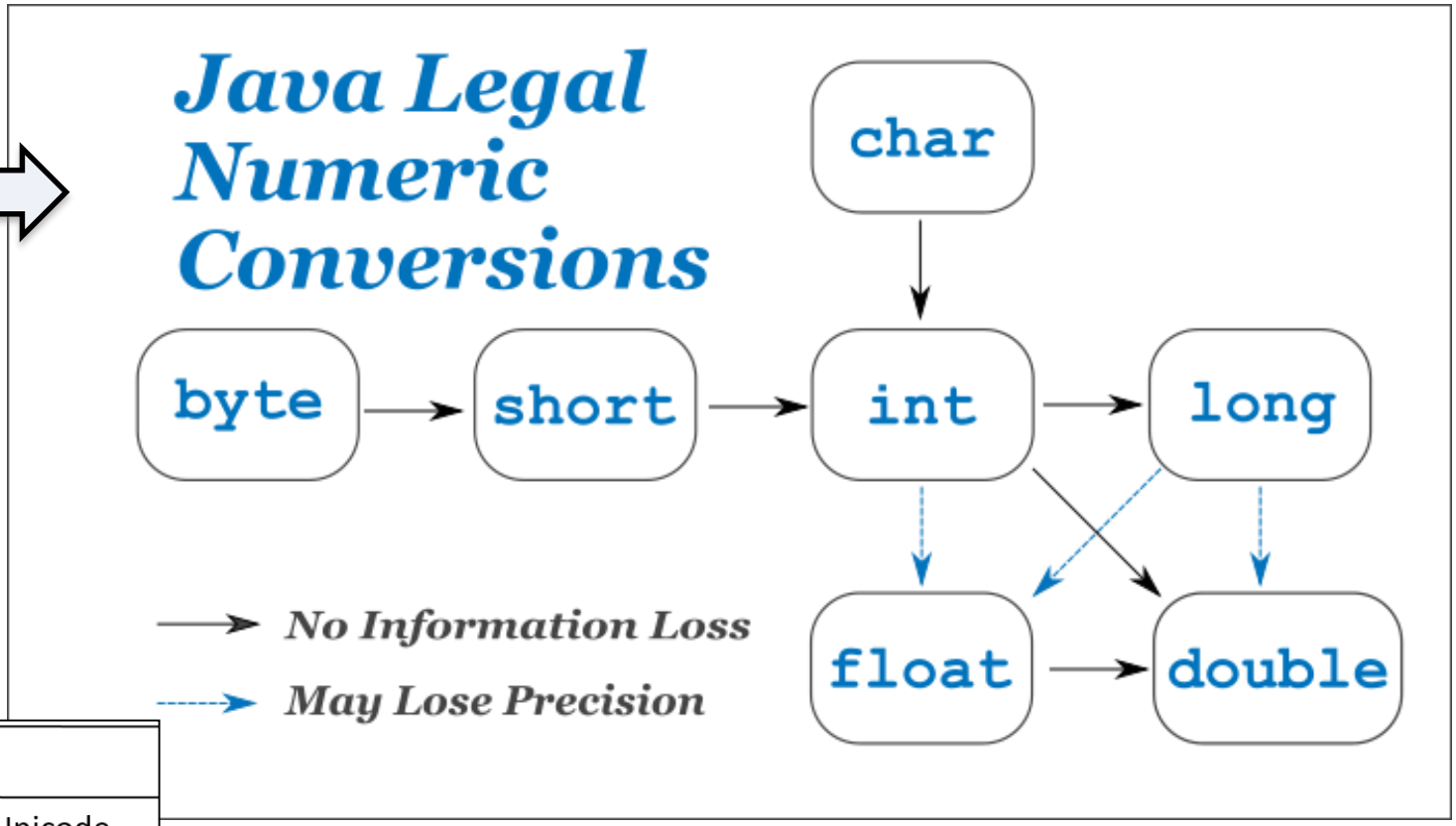
```
12 + 24.56  //int + double = double

            //lesser type is promoted to a
            //greater type, then the operation
            //is performed
```

```
int i = 12;
double d = 23.4;
i = d;      //a greater type cannot be
            //promoted to a lesser type.
            //Type mismatch and loss of
            //precision.
```

short	16-bit
char	16-bit Unicode
int	32-bit
float	32-bit
double	64-bit
long	64-bit

Type Casting

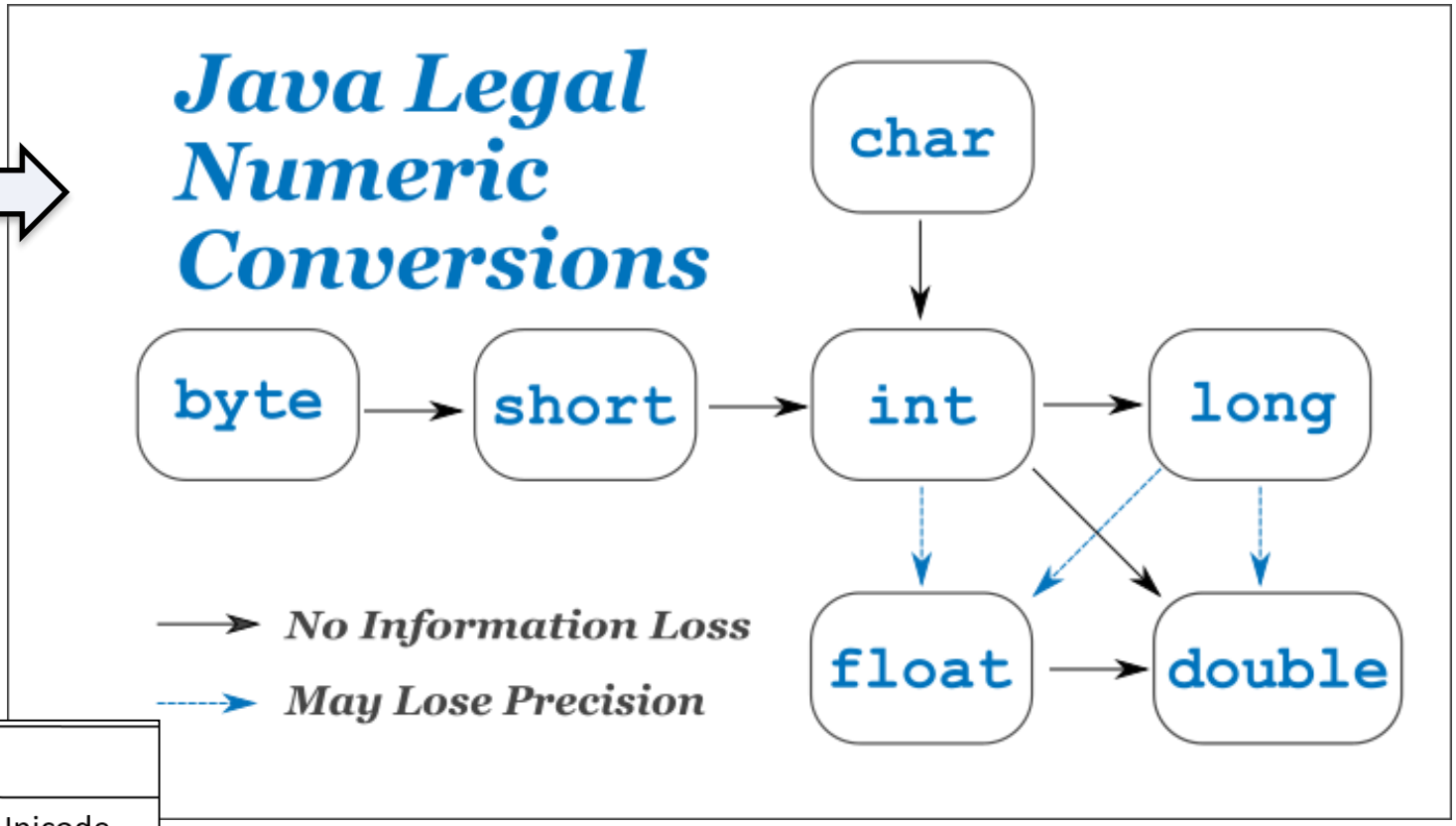
Implicit casting



short	16-bit
char	16-bit Unicode
int	32-bit
float	32-bit
double	64-bit
long	64-bit

Type Casting

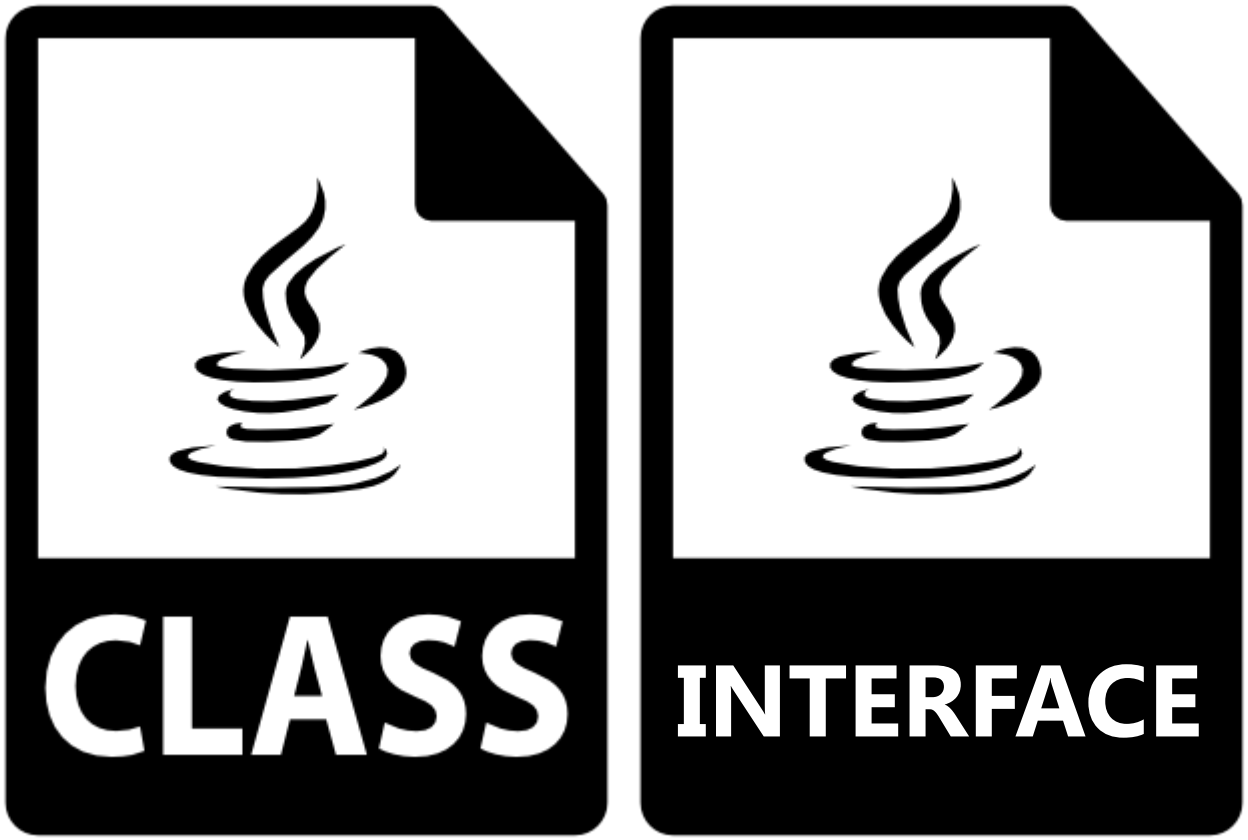
Implicit casting



short	16-bit
char	16-bit Unicode
int	32-bit
float	32-bit
double	64-bit
long	64-bit

```
//compiler error - type mismatch
int i = 34.5;
//explicit type casting
int i = (int)34.5;
```

Reference Types (aka Object Types)



Reference Types (aka Object Types)

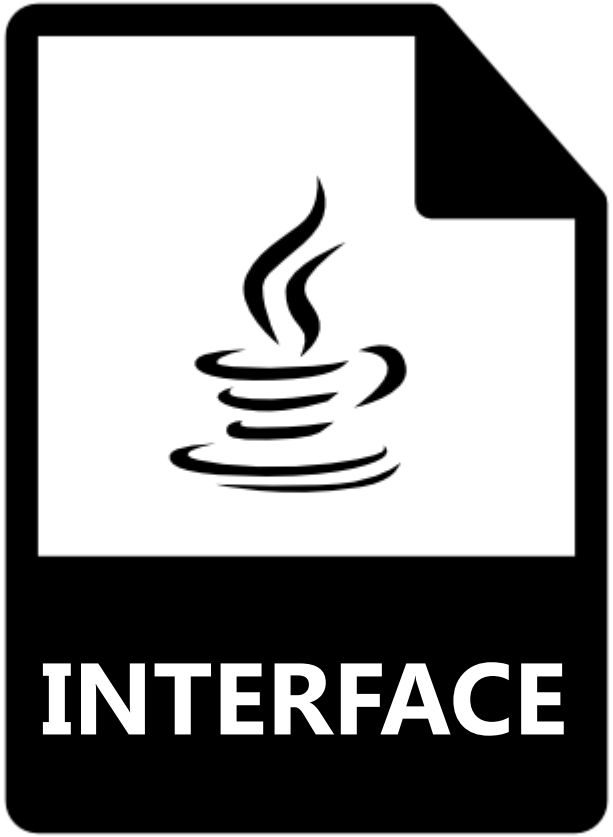
If a variable is declared as a type of class:

- An instance of that class can be assigned to it.
- An instance of any subclass of that class can be assigned to it.

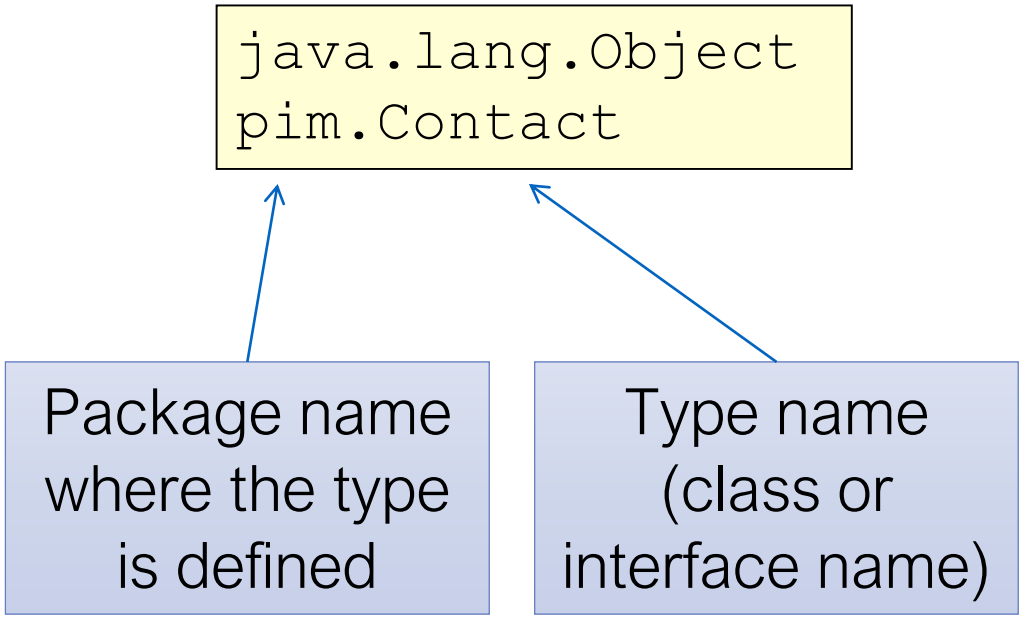
Reference Types (aka Object Types)

If a variable is declared as a type of interface:

•An instance of any class that implements the interface can be assigned to it.



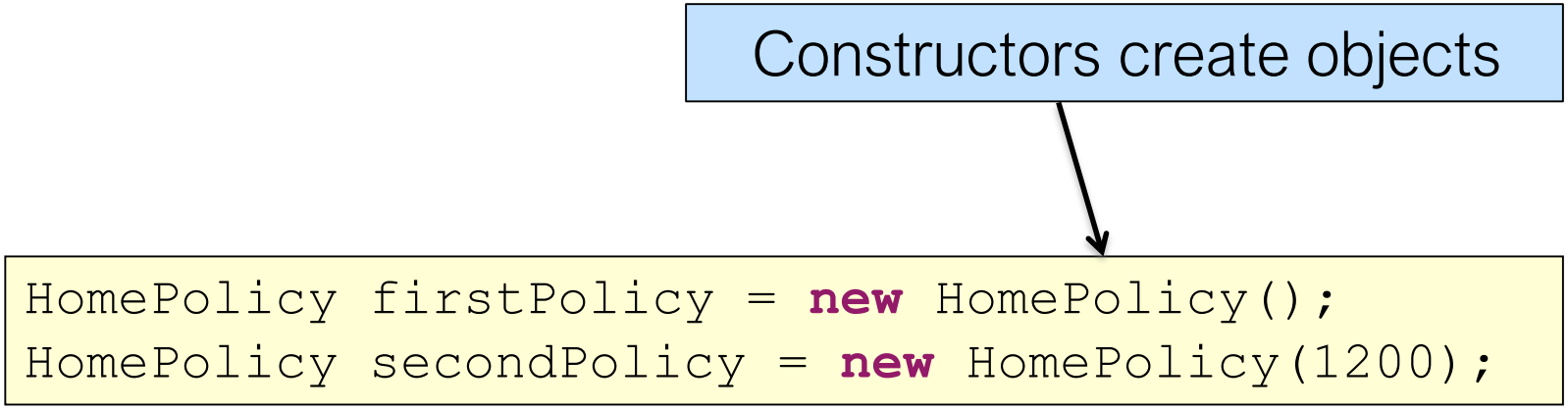
Uniquely Identifying a Reference Type



Object Operators

Keyword	Description
instanceof	object type
!=	not identical
==	identical
=	assignment

Creating Objects in Java



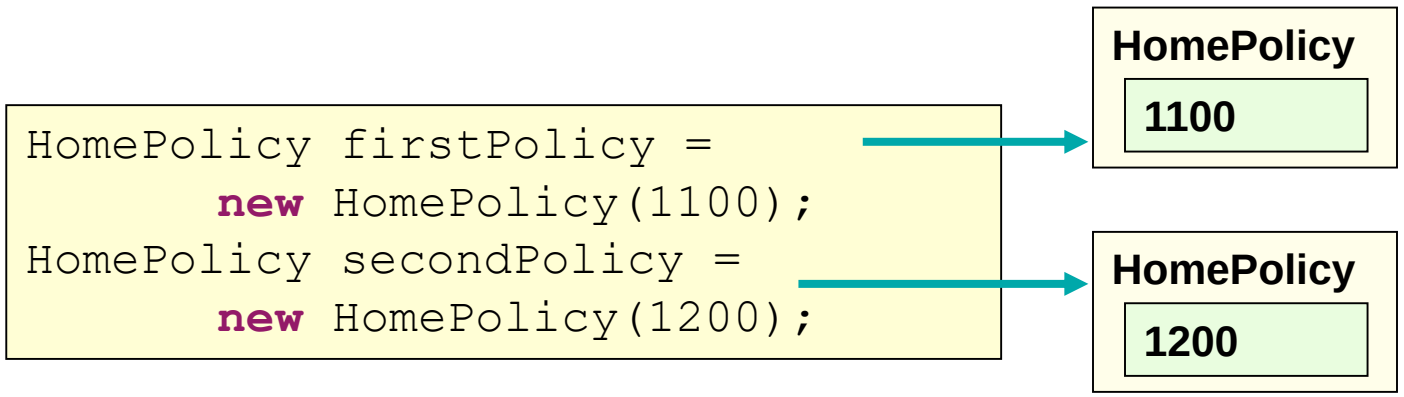
Creating Objects in Java

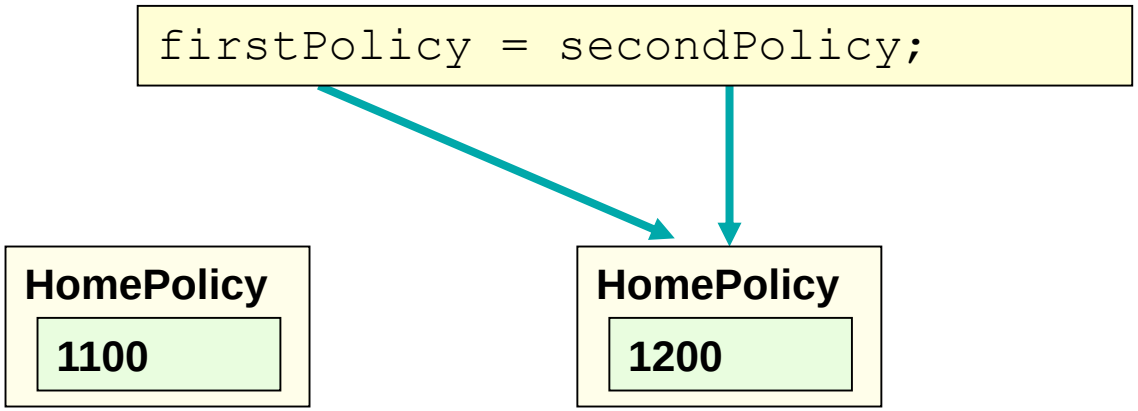
```
public class HomePolicy {

	private double premium;

	//default constructor
	public HomePolicy(){
		premium = 1000;
	}

	//overloaded constructor
	public HomePolicy(double premium){
		this.premium = premium;
	}
}
```

Constructors create objects

```
HomePolicy firstPolicy = new HomePolicy();
HomePolicy secondPolicy = new HomePolicy(1200);
```

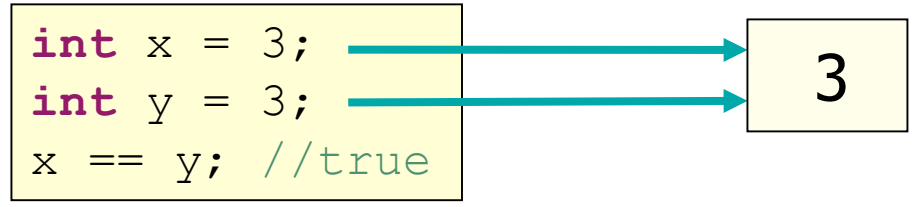
Assignment


HomePolicy firstPolicy =
new HomePolicy(1100);
HomePolicy secondPolicy =
new HomePolicy(1200);
HomePolicy
1100
HomePolicy
1200


firstPolicy = secondPolicy;
HomePolicy
1100
HomePolicy
1200

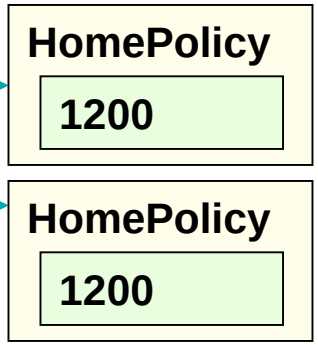
Identical Objects…

```
int x = 3;
int y = 3;
x == y; //true
```



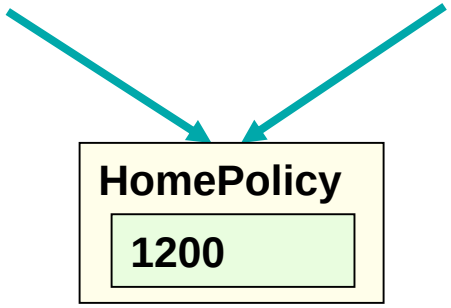
Operand == is used for checking if two objects are identical (occupy same memory space).

```
HomePolicy firstPolicy =
        new HomePolicy(1200);
HomePolicy secondPolicy =
        new HomePolicy(1200);

firstPolicy == secondPolicy;      //false
```



…Identical Objects

```
HomePolicy firstPolicy = new HomePolicy(1200);

HomePolicy secondPolicy = firstPolicy;

firstPolicy == secondPolicy;  //true
```



Reference type variables are identical if they refer to exactly the same instance of the class i.e. their memory address is the same.

Equal Objects

```
HomePolicy firstPolicy = new HomePolicy(1200,1);
HomePolicy secondPolicy = new HomePolicy(1200,1);

firstPolicy.equals(secondPolicy);
```

Equality determined by implementation of the **equals()** method…we can write this method however we please!

null

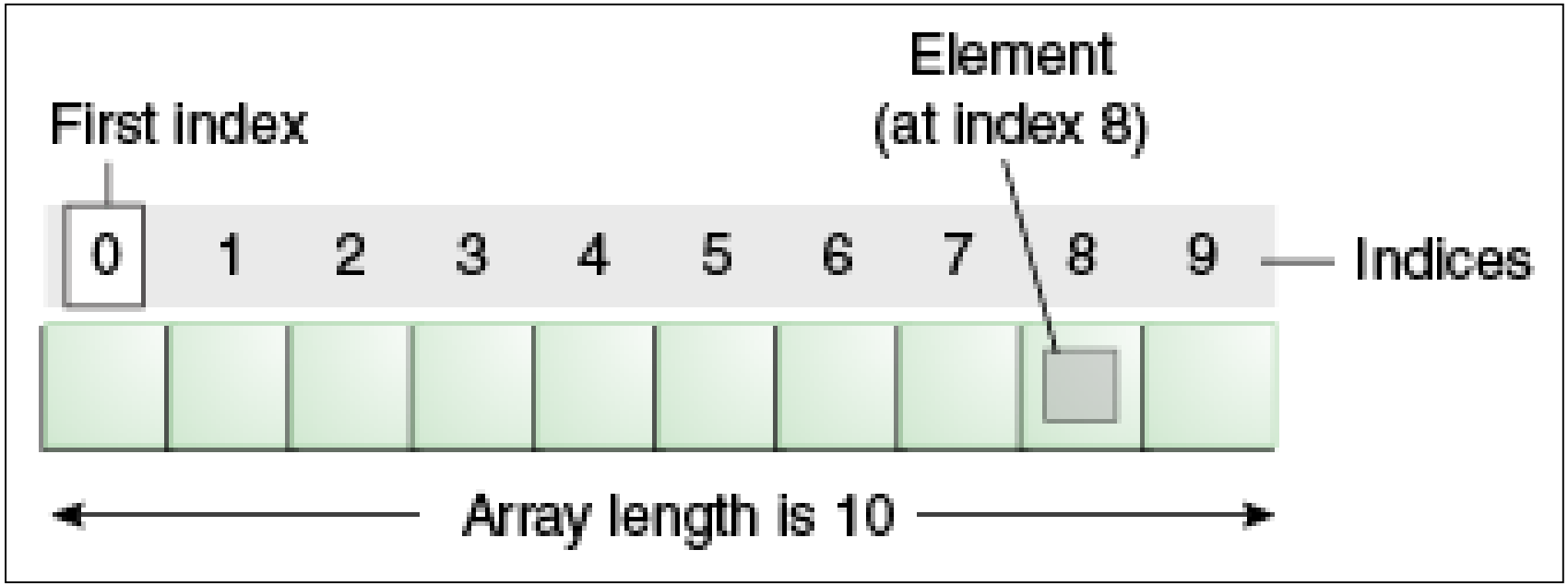
```
String one = "One";

one = null;  //used here to un-assign an object
             //from a variable

one = "1";
```

```
HomePolicy policy;  //when a variable of an
                    //object type is declared,
                    //it is assigned a value of
                    //null.
policy = new HomePolicy(1200);
…
if (policy != null)
{
  System.out.println(policy.toString());
}
```

Overview: Road Map

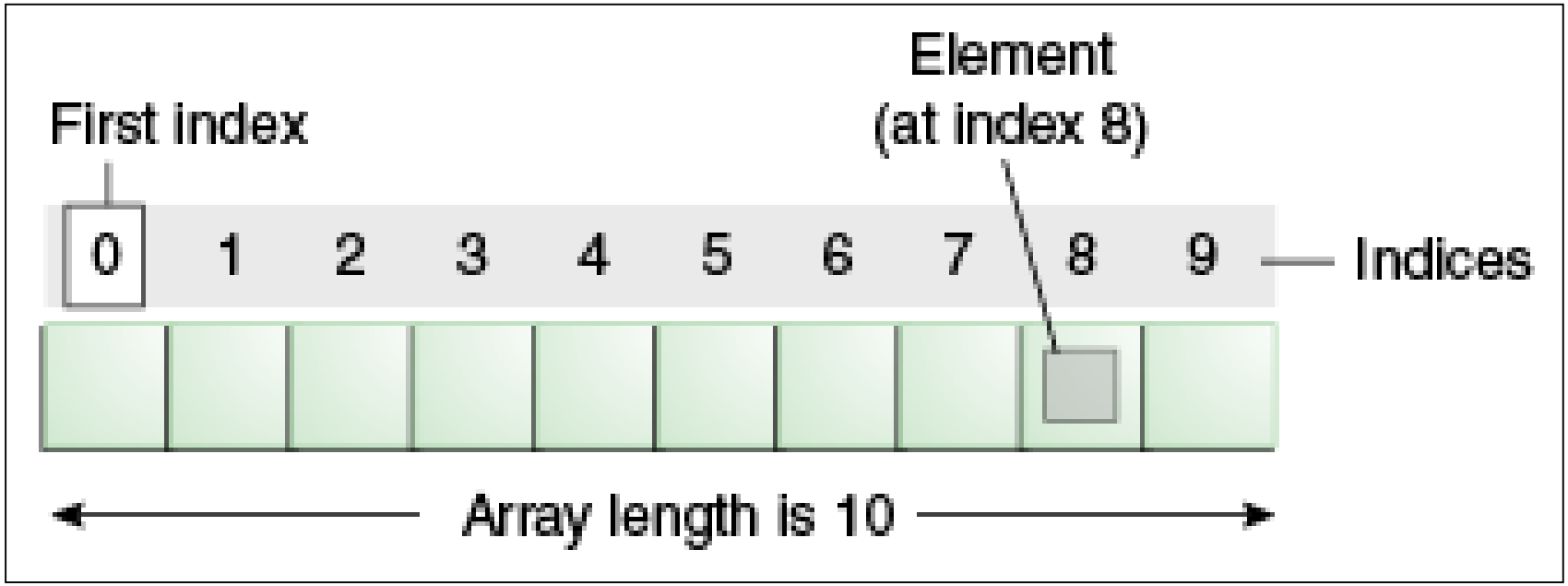
- Java Introduction
  - History
  - Portability
  - Compiler
  - Java Virtual Machine
  - Garbage collection
- Java Syntax
  - Identifiers
  - Expressions
  - Comments
- Java Basics
  - Java types
  - Primitives
  - Objects
  - Variables
  - Operators
  - Identity and equality
- Arrays
  - What are arrays?
  - Creating arrays
  - Using arrays

What is an Array?



Fixed-size collection containing elements of the same type (either primitives or objects)

What is an Array?



Fixed-size collection containing elements of the same type (either primitives or objects)

Automatically bounds-checked → exception thrown if you access an element that doesn't exist.

Declaring Arrays

[] indicates that a variable references an array (of one particular data type)

```
int[] arrayOfIntegers;
String[] arrayOfStrings;
```

or

```
int arrayOfIntegers[];
String arrayOfStrings[];
```

Creating Arrays Explicitly

```
int arrayOfIntegers[];
arrayOfIntegers = new int[5];
```

One approach:

- use the keyword **new** to create an array.
- You must specify the size of the array here.
- Elements are initialized to default values.

Creating and Initializing an Array

Another approach:

- create and initialize an array using curly brackets.
- Can only be done when declaring a variable.

```
int[] arrayOfIntegers = {1,2,3,4,5};
```

```
int[] arrayOfIntegers;
arrayOfIntegers = {1,2,3,4,5};  //compile error
```

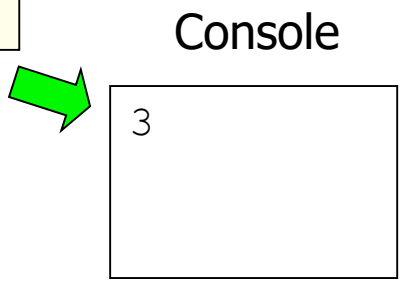
Initializing Arrays

- If not using initializer, an array can be initialized by storing elements at a specific index location.

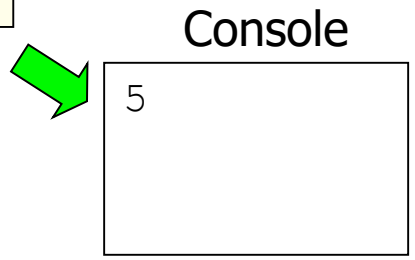
```
int[] arrayOfIntegers;
arrayOfIntegers = new int[5];
arrayOfIntegers[0] = 1;
arrayOfIntegers[1] = 2;
arrayOfIntegers[2] = 3;
arrayOfIntegers[3] = 4;
arrayOfIntegers[4] = 5;
```

Manipulating Arrays

```
int[] arrayOfIntegers = {1,2,3,4,5};
System.out.println(arrayOfIntegers[2]);
```




```
int[] arrayOfIntegers = {1,2,3,4,5};
System.out.println(arrayOfIntegers.length);
```



Multi-Dimensional Arrays

Matrix:
2 rows, 5 columns

```
int[][] arrayOfIntegers = new int[2][5];
```

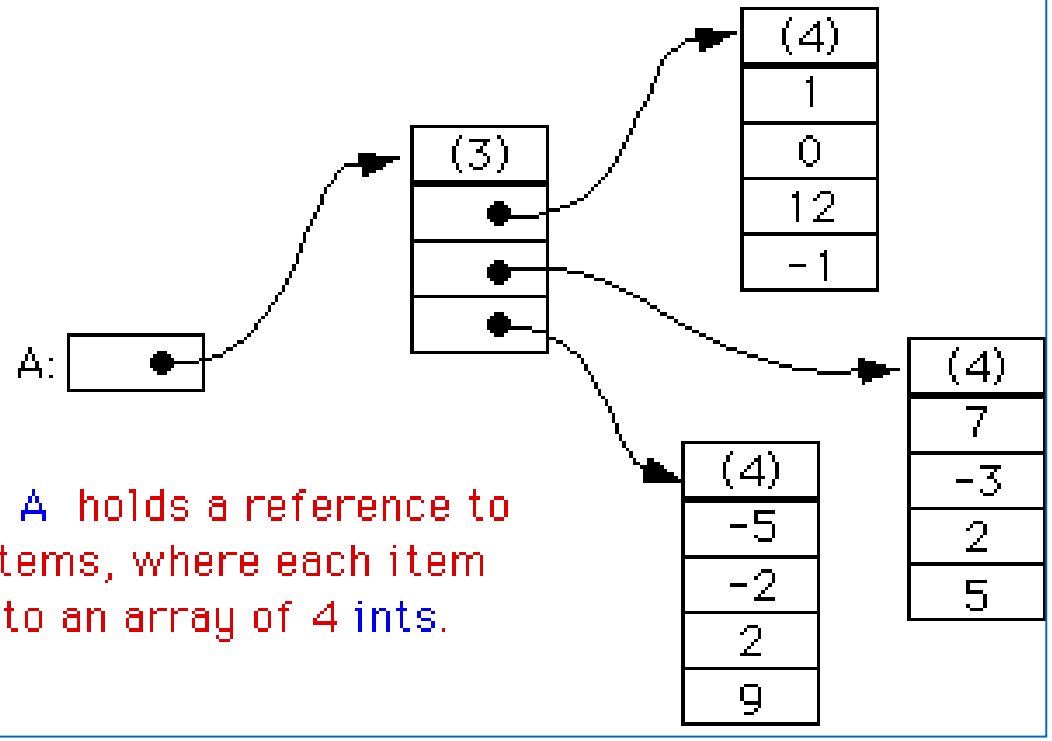
Arrays can be 2-dimensional, 3-dimensional….n-dimensional

Multi-Dimensional Arrays

If you create an array A = new int[3][4], you should think of it as a "matrix" with 3 rows and 4 columns.

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	9

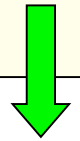


But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.

Manipulating Multi-Dimensional Arrays

- Using array initializers

```
int[][] arrayOfIntegers = {{1,2,3,4,5},{6,7,8,9,10}};

System.out.println(arrayOfIntegers[1][2]);
```



```
8
```

What we covered in this lecture:

- **Overview**
  - Introduction
  - Syntax
  - Basics
  - Arrays
- **Classes**
  - Classes Structure
  - Static Members
  - Commonly used Classes
- **Control Statements**
  - Control Statement Types
  - If, else, switch
  - For, while, do-while

- **Inheritance**
  - Class hierarchies
  - Method lookup in Java
  - Use of this and super
  - Constructors and inheritance
  - Abstract classes and methods

  Interfaces
- **Collections**
  - ArrayList
  - HashMap
  - Iterator
  - Vector
  - Enumeration
  - Hashtable

- **Exceptions**
  - Exception types
  - Exception Hierarchy
  - Catching exceptions
  - Throwing exceptions
  - Defining exceptions

  Common exceptions and errors
- **Streams**
  - Stream types
  - Character streams
  - Byte streams
  - Filter streams
  - Object Serialization

Any questions?