Java Overview

Introduction to the Java Programming Language

Produced by

Eamonn de Leastar
edeleastar@wit.ie

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit


Unearthing the Excellence in Java
Java
The Good Parts
O'REILLY®
Jim Waldo

Essential Java

- **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

Overview

- 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

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 History

- Originally was called the “Oak”
- Was intended to be used in consumer electronics
  - Platform independence was one of the requirements
  - Based on C++, with influences from
    other OO languages (Smalltalk, Eiffel…)
- Started gaining popularity in 1995
  - Renamed to “Java”
  - Was good fit for the Internet applications

Portability

- Java is platform independent language
  - Java code can run on any platform
  - Promotes the idea of writing the code on one platform and running it on any other
- Java also supports *native methods*
  - Native methods are platform specific
  - Breaks the idea of platform independence

Compiler

- Java source code is stored in .java files
- Compiler compiles code into .class files
  - The compiled code is the bytecode that can run on any platform
  - Bytecode is what makes Java platform independent
- Bytecode is not a machine code
  - The code must be interpreted in the machine code at runtime

Java Virtual Machine (JVM)

- Platform specific
  - Processes bytecode at the runtime by translating bytecode into machine code
  - This means that Java is interpreted language
  - JVM is different for different platforms and bytecode is the same for different platforms

Memory Management

- Automatic garbage collection is built in the language
  - No explicit memory management is required
  - Occurs whenever memory is required
  - Can be forced programmatically
  - Garbage collector frees memory from objects that are no longer in use

Distributed Systems

- Java provides low level networking
  - TCP/IP support, HTTP and sockets
- Java also provides higher level networking
  - Remote Method Invocation (RMI) is Java's distributed protocol
    - Used for communication between objects that reside in different Virtual Machines
    - Commonly used in J2EE (Java 2 Enterprise Edition) Application Server
  - ~~CORBA could also be used with Java~~

Concurrency

- Java includes support for multithreaded applications
  - API for thread management is part of the language
- Multithreading means that various processes/tasks can run concurrently in the application
- Multithreaded applications may increase:
  - Availability
  - Asynchronization
  - Parallelism

Road Map

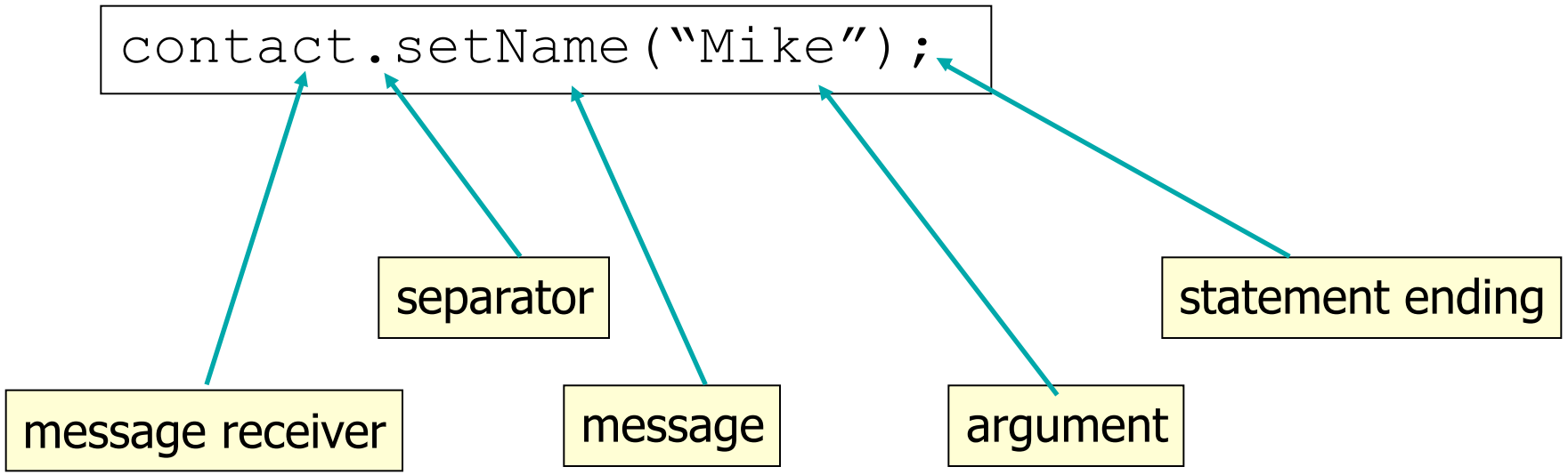
- Java Introduction
  - Background
  - Portability
  - Compiler
  - Java Virtual Machine
  - Garbage collection
- Java Syntax
  - Identifiers
  - Expressions
  - Comments
- Java Basics
  - Java types
  - Primitives
  - Objects
  - Variables
  - Operators
  - Identity and equality
- Arrays
  - What are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

Identifiers

- Used for naming classes, interfaces, methods, variables, fields, parameters
- Can contain letters, digits, underscores or dollar-signs
- There are some rules that apply:
  - First character in the identifier cannot be a digit
    - Can be a letter, underscore or dollar sign
  - Literals true, false and null cannot be used
  - Reserved words cannot be used

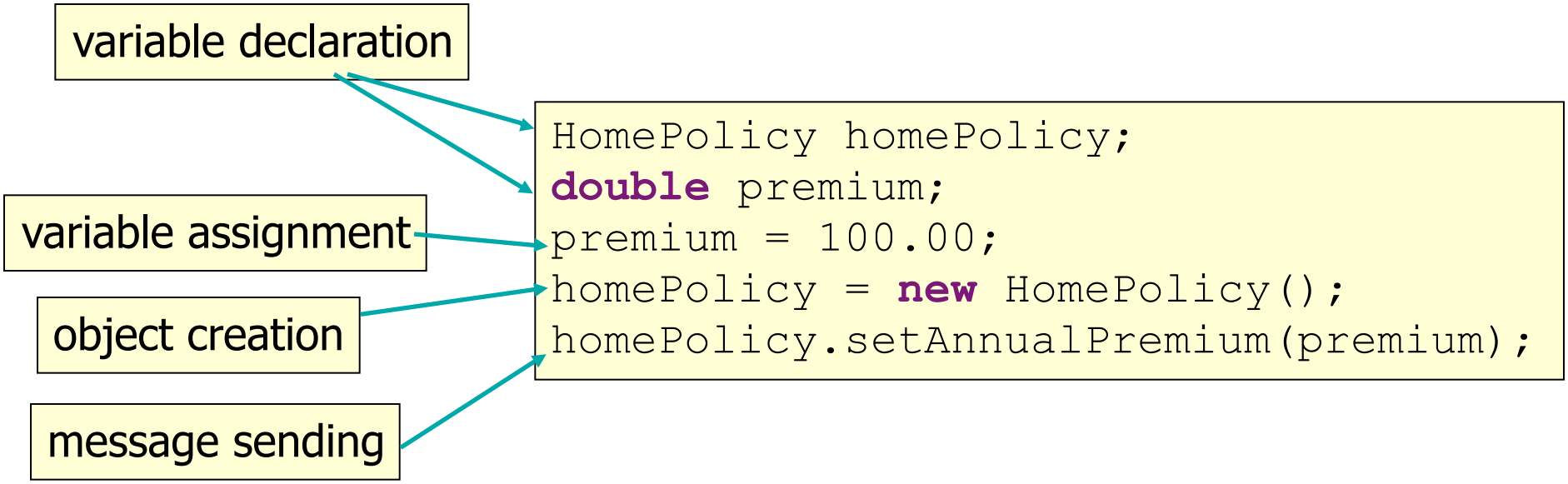
Messages and Objects

- Objects send messages to other objects



Expressions

- Statements are the basic Java expressions
  - Semicolon (;) indicates end of a statement



Empty Expression

- Semicolon on its own in the line
  - Can be used to indicate *do nothing* scenario in the code

```
;      //this is an empty statement
```

```
for(int i=1; i<3; i++) ;
      System.out.println(i);
```

  - We would expect the code to print 0,1,2 but it prints only 0 because of the empty statement

Comments

- 3 different types of comments in Java:
  - Single line comment
    - Starts with // and ends at the end of the line
  - Multiple line comment
    - Starts with /* and ends with */
  - Javadoc comment
    - Starts with /** and ends with */
    - Used by Javadoc program for generating Java documentation

```
/** Javadoc example comment.
 *  Used for generation of the documentation.
 */

/*  Multiple line comment.
 *
 */

//  Single line comment.
```

Literals

- Represent hardcoded values that do not change
- Typical example are string literals
  - When used compiler creates an instance of String class

```
String one = "One";
String two = "Two";
```

=

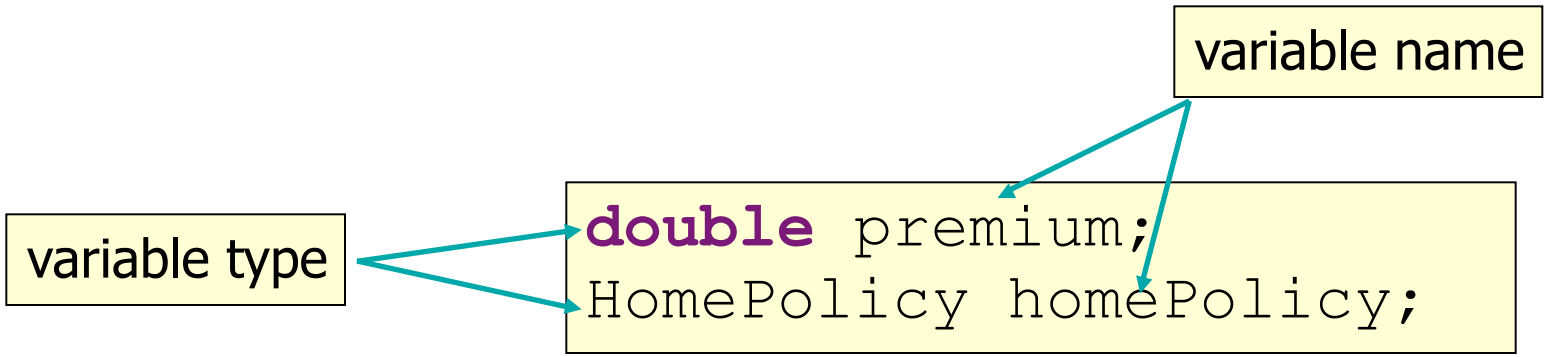
```
String one = new String("One");
String two = new String("Two");
```

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

- There are two different types in Java:
  - Primitive data type
  - Reference type
- Java is strongly typed language
  - Fields, variables, method parameters and returns must have a type



Primitives

- Primitives represent simple data in Java
- Primitives are not objects in Java
  - Messages cannot be sent to primitives
  - Messages can be sent to other Java objects that represents primitives
  - These are known as wrapper objects in Java (such as Double, Integer, and Boolean)

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

Primitives 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**		

boolean Type

- Commonly used in control statements
- Consists of two boolean literals:
  - true
  - false

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

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

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

char Type

- Represents characters in Java
- Uses 16-bit unicode for support of internationalization
- Character literals appear in single quotes, and include:
  - Typed characters, e.g. `'z'`
  - Unicode, e.g. `'\u0040'`, equal to `'@'`
  - Escape sequence, e.g. `'\n'`

Escape Sequence Characters

- Commonly used with print statements

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

Numeric Types

- There are generally two different types:
  - Integer: byte, short, int, and long
  - Floating-point: float, and double
- Literals can be used for all but byte and short types
  - An int is converted to byte if it fits to 8-bits
  - An int is converted to short if it fits to 16-bits

```
12      //decimal integer 12
12L     //long decimal 12
0x1E    //hexadecimal integer 30
23.f    //float
30.7    //double
```

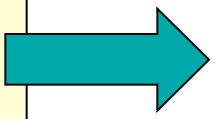
Manipulating Numeric Types

- A lesser type is promoted to greater type and than operation is performed

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

- A greater type cannot be promoted to lesser type
  - Assigning double value to int type variable would result in compile error

```
int i = 12;
double d = 23.4;
i = d;
```



Type mismatch

Type Casting

- Values of greater precision cannot be assigned to variables declared as of lower precision types
- Type casting makes primitives to change their type
  - Used to assign values of greater precision to variables declared as lower precision
    - e.g. it's possible to type cast double to int type

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

Reference Types…

- Reference types in Java are class or interface
  - They are also known as 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
- If a variable is declared as a type of interface
  - An instance of any class that implements the interface can be assigned to it

…Reference Type

- Reference type names are uniquely identified by:
  - Name of the package where type is defined (class or interface)
  - Type name

```
java.lang.Object
pim.Contact
```

Object Operators

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

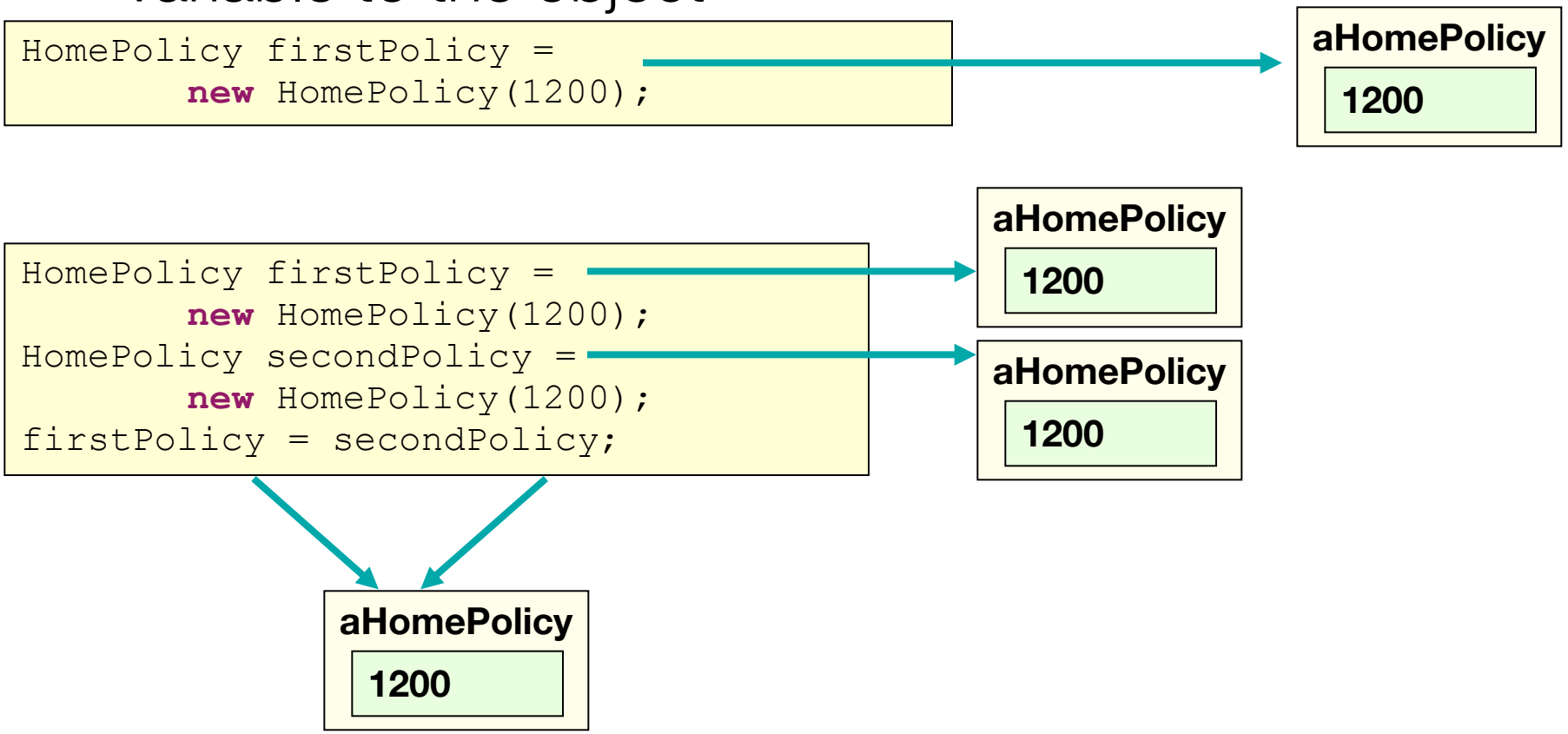
Creating Objects in Java

- Objects are, in Java, created by using constructors
- Constructors are methods that have same name as the class
  - They may accept arguments mainly used for fields initialization
  - If constructor is not defined, the default constructor is used

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

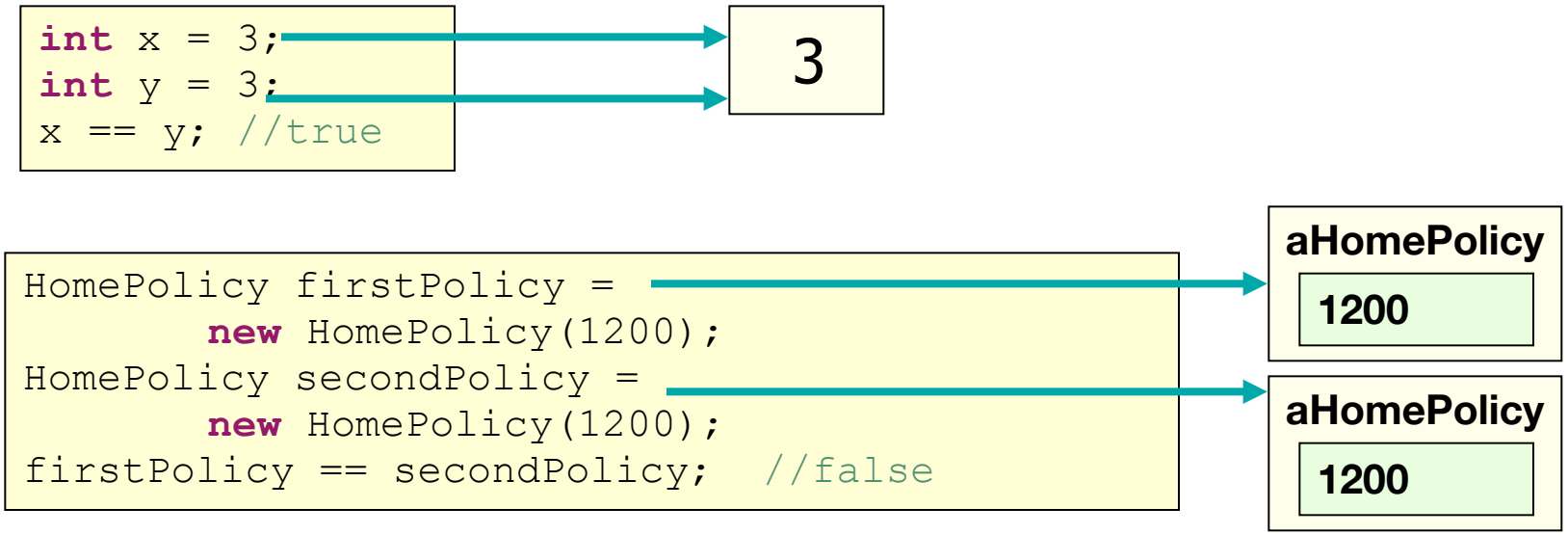
Assignment

- Assigning an object to a variable binds the variable to the object



Identical Objects…

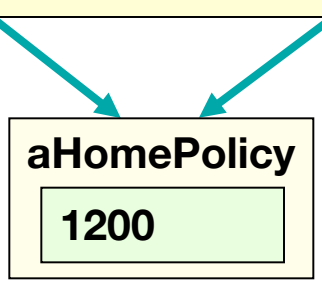
- Operand == is used for checking if two objects are identical
  - Objects are identical if they occupy same memory space



…Identical Objects

- Variables that reference objects are compared by value
  - Objects are identical if their memory addresses are the same
- Variables are identical if they refer to exactly same instance of the class

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



Equal Objects

- Determined by implementation of the equals() method
  - Default implementation is in the Object class and uses == (identity)
  - Usually overridden in subclasses to provide criteria for equality

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

null

- Used to un-assign object from a variable
  - Object is automatically garbage collected if it does not have references
- When a variable of object type is declared it is assigned null as a value

```
String one = "One";
one = null;
one = "1";
```

```
HomePolicy policy;
policy = new HomePolicy(1200);
…
if (policy != null)
{
  System.out.println(policy.toString());
}
```

Road Map

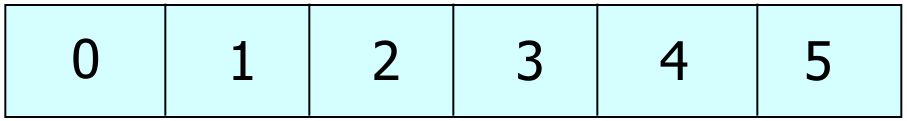
- Arrays
  - What are arrays?
  - Creating arrays
  - Using arrays

What is an Array?

- Arrays are basic collections in Java
  - They contain elements of the same type
  - Elements can either be Java objects or primitives
- Arrays are fixed-size sequential collection
  - Size is predefined, and arrays cannot grow
- Arrays are objects

Array Basics

- The first element in array is at the zero index



- Arrays are automatically bounds-checked
  - When accessing elements that are out of bounds, an exception will be thrown
  - For example, accessing element at index 6 in the above example will throw the exception

Creating Arrays…

- Arrays store objects of specific type
  - One array cannot store objects of different types, String and int for example
- To define a variable that holds an array, you suffix the type with square brackets []
  - This indicates that variable references an array

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

…Creating Arrays…

- Alternative ways to define an array include:
  - Suffixing variable name with brackets

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

…Creating Arrays

- There are two ways to create an array:
  - Explicitly using the keyword new
  - Using array initializer
- When creating an array explicitly its size must be specified
  - This indicates desired number of elements in the array
  - Elements in the array are initialized to default values

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

Array Initializer

- Used for creating and initializing arrays
  - Array elements are initialized within the curly brackets

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

- Can only be used when declaring variable
  - Using array initializer in a separate step will result in a compilation error

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

Initializing Arrays

- If not using initializer, an array can be initialized by storing elements at proper index

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

Manipulating Arrays

- An element of the array is accessed by accessing index at which element is stored

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

Console

```
3
```

- An array size can be obtained by asking for its length
  - Used commonly in control statements (loops)

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

Console

```
5
```

Multi-Dimensional Arrays

- An array can contain elements of other arrays
  - Such an array is known as multi-dimensional array
  - There is no limit is number of dimensions
    - Arrays can be 2-dimensional, 3-dimensional, and n-dimensional

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

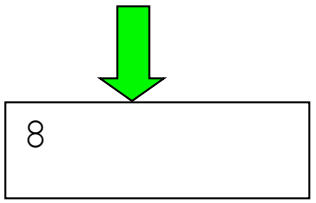
Manipulating Multi-Dimensional Arrays

- Multi-dimensional arrays are created like any other arrays
  - Using the keyword new
  - Using array initializers

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

- Elements in multi-dimensional array are also accessed using their indices

```
int[][] arrayOfIntegers = {{1,2,3,4,5},{6,7,8,9,10}};
System.out.println(arrayOfIntegers[1][2]);
```



Summary

- Java Introduction
  - Background
  - 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