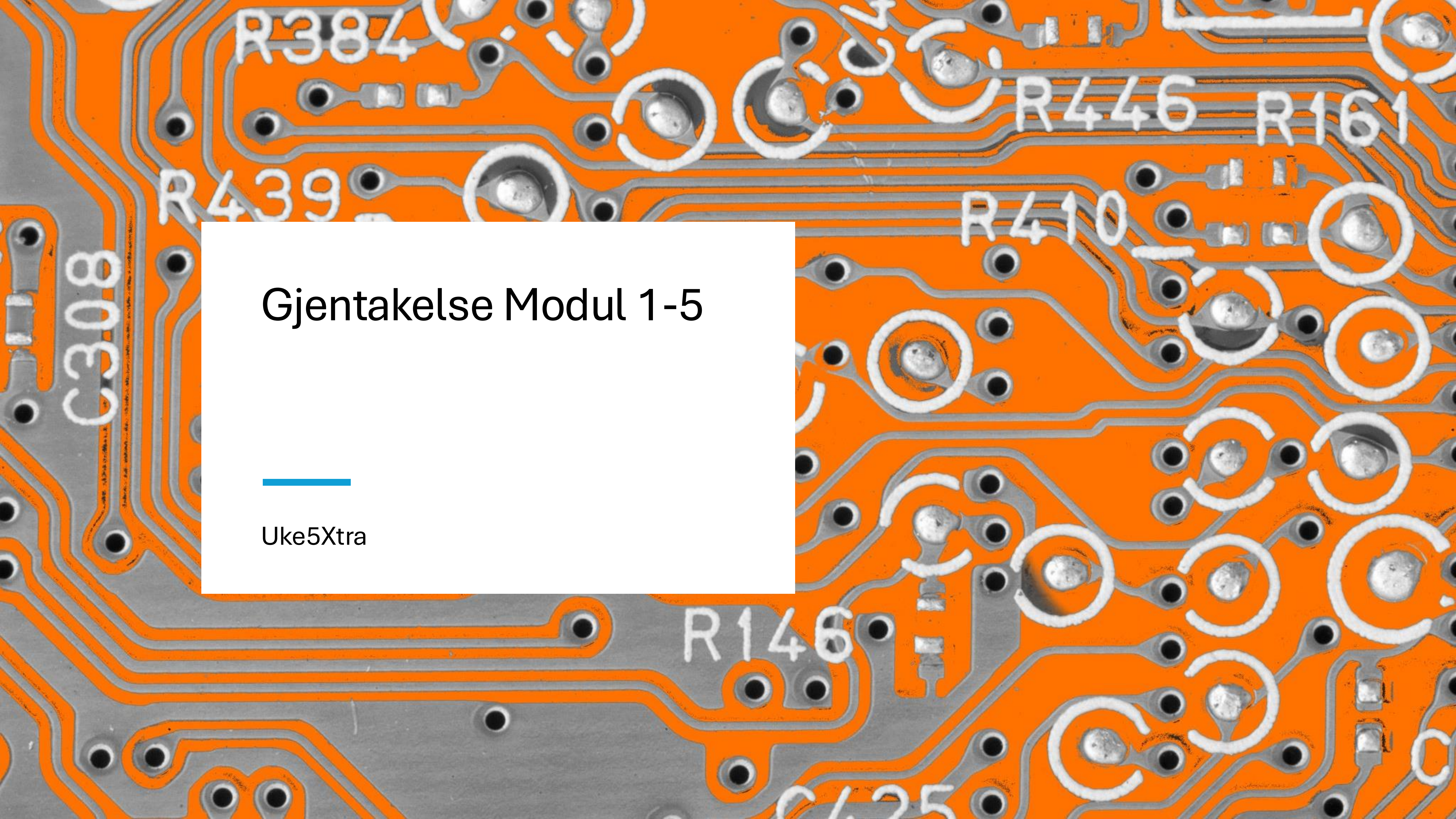
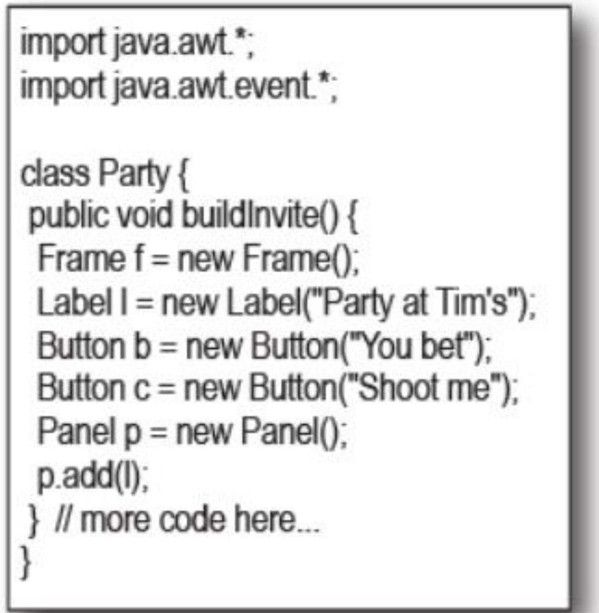


Gjentakelse Modul 1-5

Uke5Xtra



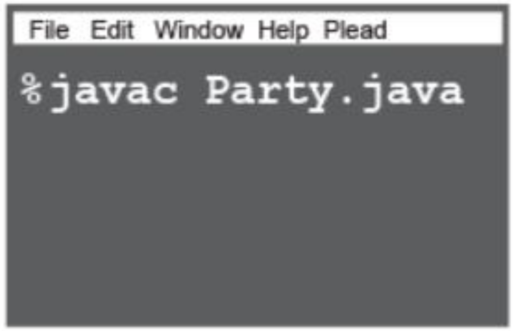
```
import java.awt.*;
import java.awt.event.*;

class Party {
 public void buildInvite() {
  Frame f = new Frame();
  Label l = new Label("Party at Tim's");
  Button b = new Button("You bet");
  Button c = new Button("Shoot me");
  Panel p = new Panel();
  p.add(l);
 } // more code here...
}
```

Source

1

Type your source code.

Save as: ***Party.java***



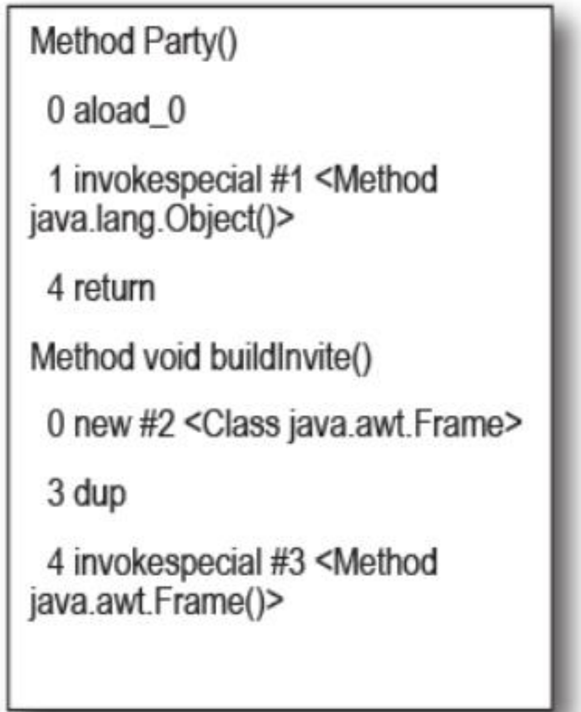
```
%javac Party.java
```

Compiler

2

Compile the ***Party.java*** file by running `javac` (the compiler application). If you don't have errors, you'll get a second document named ***Party.class***.

The compiler-generated Party.class file is made up of *bytecodes*.

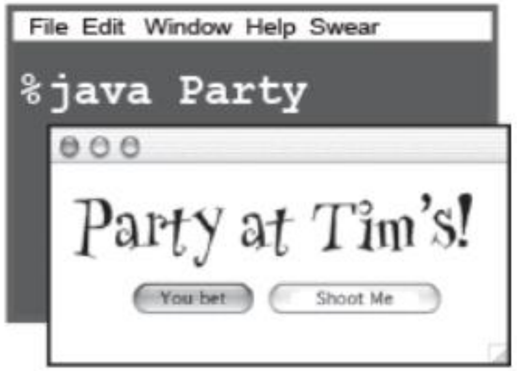


```
Method Party()

 0 aload_0

 1 invokespecial #1 <Method
java.lang.Object()>

 4 return

Method void buildInvite()

 0 new #2 <Class java.awt.Frame>

 3 dup

 4 invokespecial #3 <Method
java.awt.Frame()>
```

Output (code)

3

Compiled code: ***Party.class***



```
%java Party
```

Virtual Machines

4

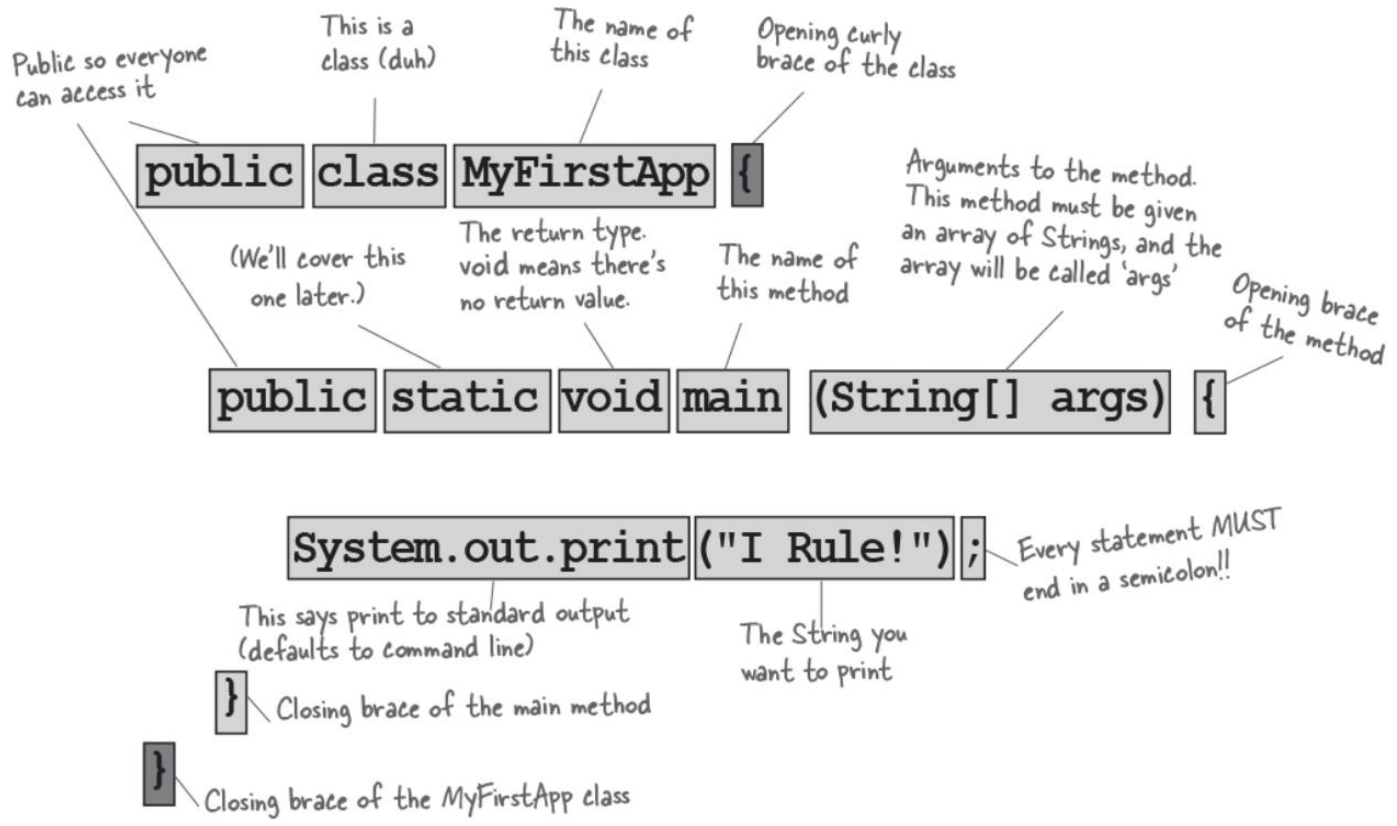
Run the program by starting the Java Virtual Machine (JVM) with the ***Party.class*** file. The JVM translates the *bytecode* into something the underlying platform understands, and runs your program.

Forskjellige nøkkelord

public / private

void / main

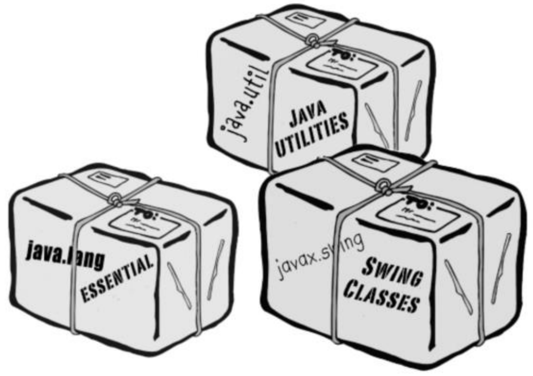
psvm / sout / souf


Public so everyone can access it
This is a class (duh)
The name of this class
Opening curly brace of the class
(We'll cover this one later.)
The return type. void means there's no return value.
The name of this method
Arguments to the method. This method must be given an array of Strings, and the array will be called 'args'
Opening brace of the method
Every statement MUST end in a semicolon!!
This says print to standard output (defaults to command line)
The String you want to print
Closing brace of the main method
Closing brace of the MyFirstApp class

```
public class MyFirstApp {
public static void main (String[] args) {
System.out.print("I Rule!");
}
}
```

You ´ll do it again..and again.. And again

Naming Conventions
camelCase
PascalCase
snake_case



```
java.util.Random randomGenerator = new java.util.Random();
int rand1 = randomGenerator.nextInt(oneLength);
```

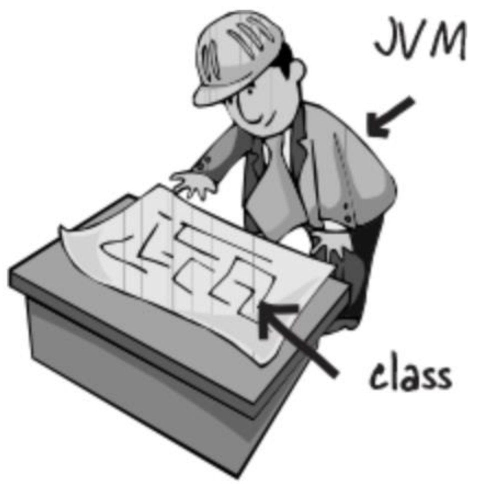
Re+CAP

- Reserved words : If, else, while, for… og mange andre

- Class, Object : PascalCase.
- variables, methods : camelCase .
- ENUM, CONSTANTS : SNAKE_CASE

- Java is a Typed language: primitive type, object types , packages

A class is not an object (but it’s used to construct them)



• [illegible]

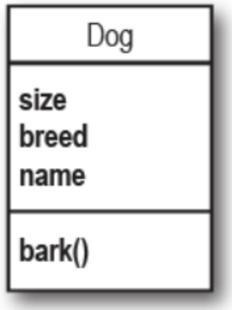
Classes & Objects

1 Write your class

```
class Dog {
  int size;
  String breed;
  String name;

  void bark() {
    System.out.println("Ruff! Ruff!");
  }
}
```

Instance variables

A method



2 Write a tester (TestDrive) class

```
class DogTestDrive {
  public static void main(String[] args) {
    // Dog test code goes  here
  }
}
```

Just a main method (we're gonna put code in it in the next step)

3 In your tester, make an object and access the object's variables and methods

```
class DogTestDrive {
  public static void main(String[] args) {
    Dog d = new Dog();
    d.size = 40;
    d.bark();
  }
}
```

Make a Dog object

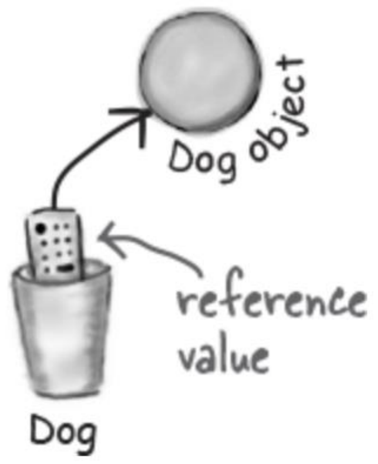
Use the dot operator (.) to set the size of the Dog and to call its bark() method

Dot operator

Test – hvem er jeg?

• Jeg hjelper til med *innkapsling :*	private, public, getter, setter
• Jeg kan endre meg under kjøring *:*	objekt, instansvariabel
• Jeg oppfører meg som en mal*:*	klasse

The Dog object itself does not go into the variable!

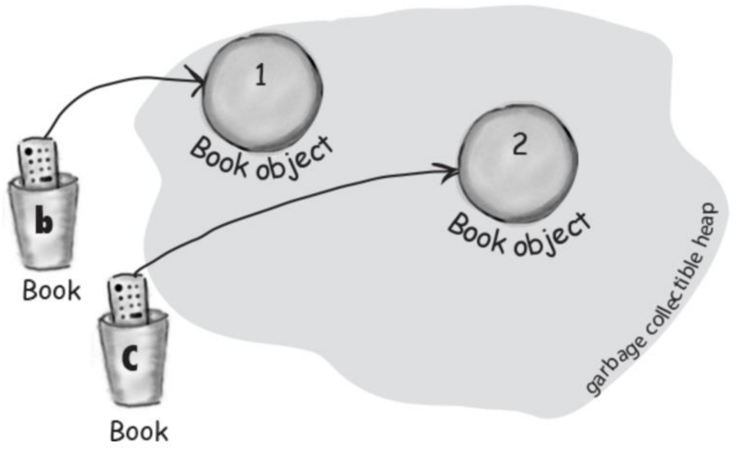


think of this
like this
WAG
BARK
EAT
PLAY
FETCH
DOG
Think of a Dog

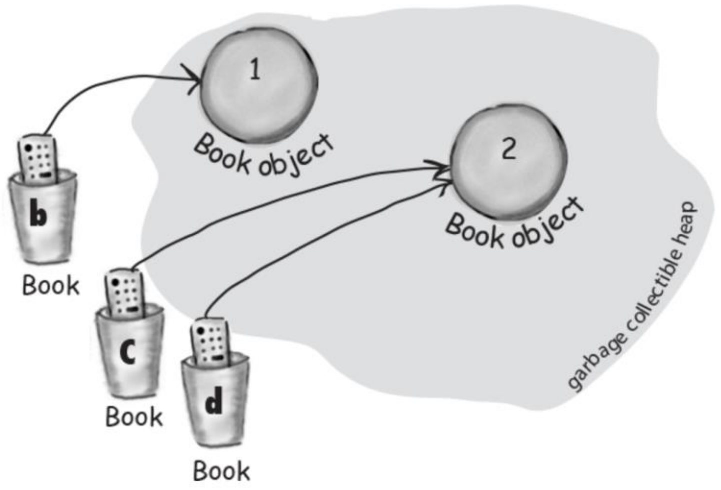
d.setBite()

Robodog d;   if (d.bark())   d.set~~Byte~~(false);

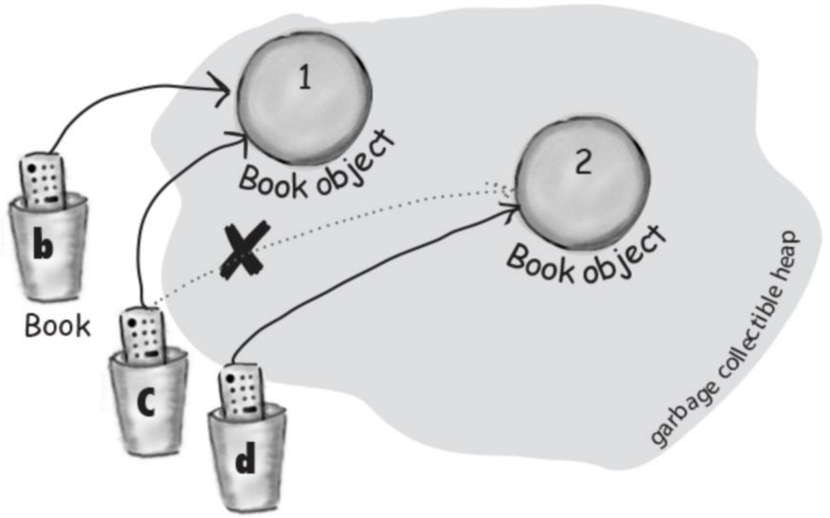
Life on the garbage-collectible heap



```
Book b = new Book();
Book c = new Book();
```

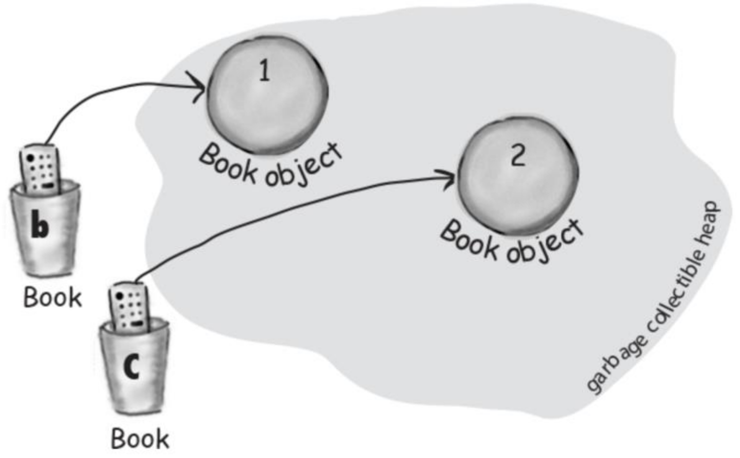


```
Book d = c;
```

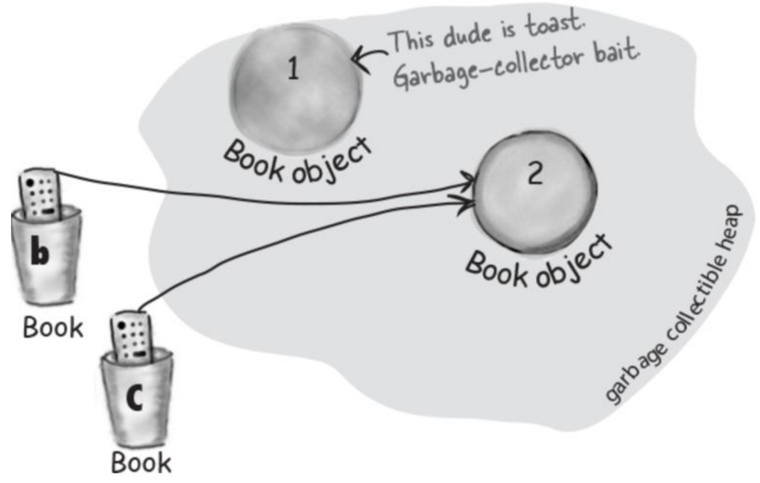


```
c = b;
```

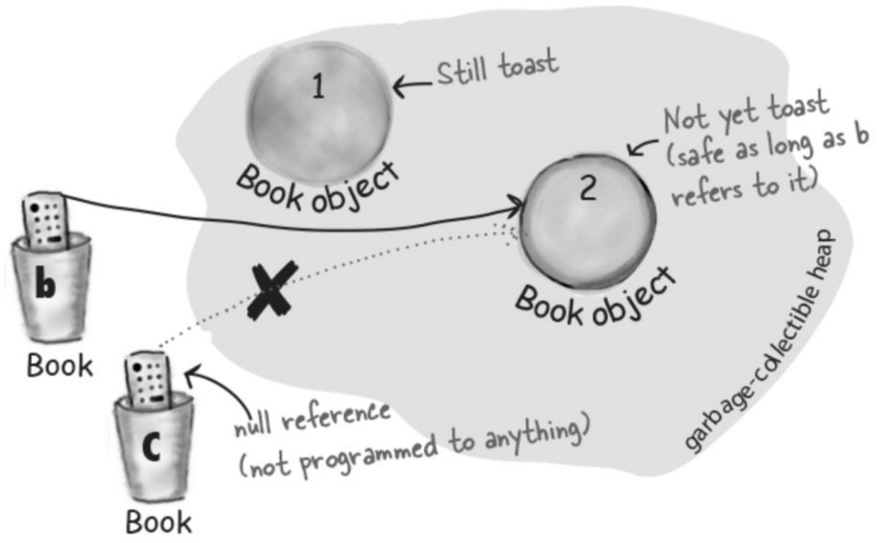
Life and death on the heap



```
Book b = new Book();
Book c = new Book();
```



```
b = c;
```



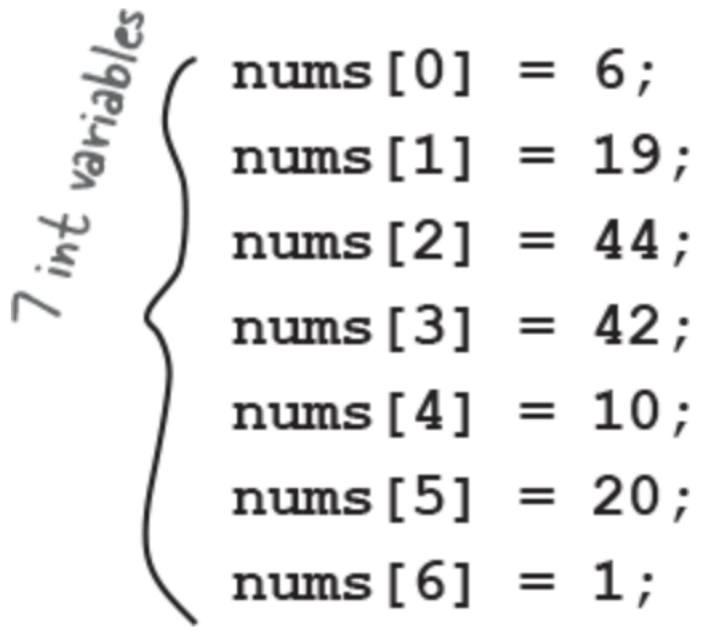
```
c = null;
```

Array

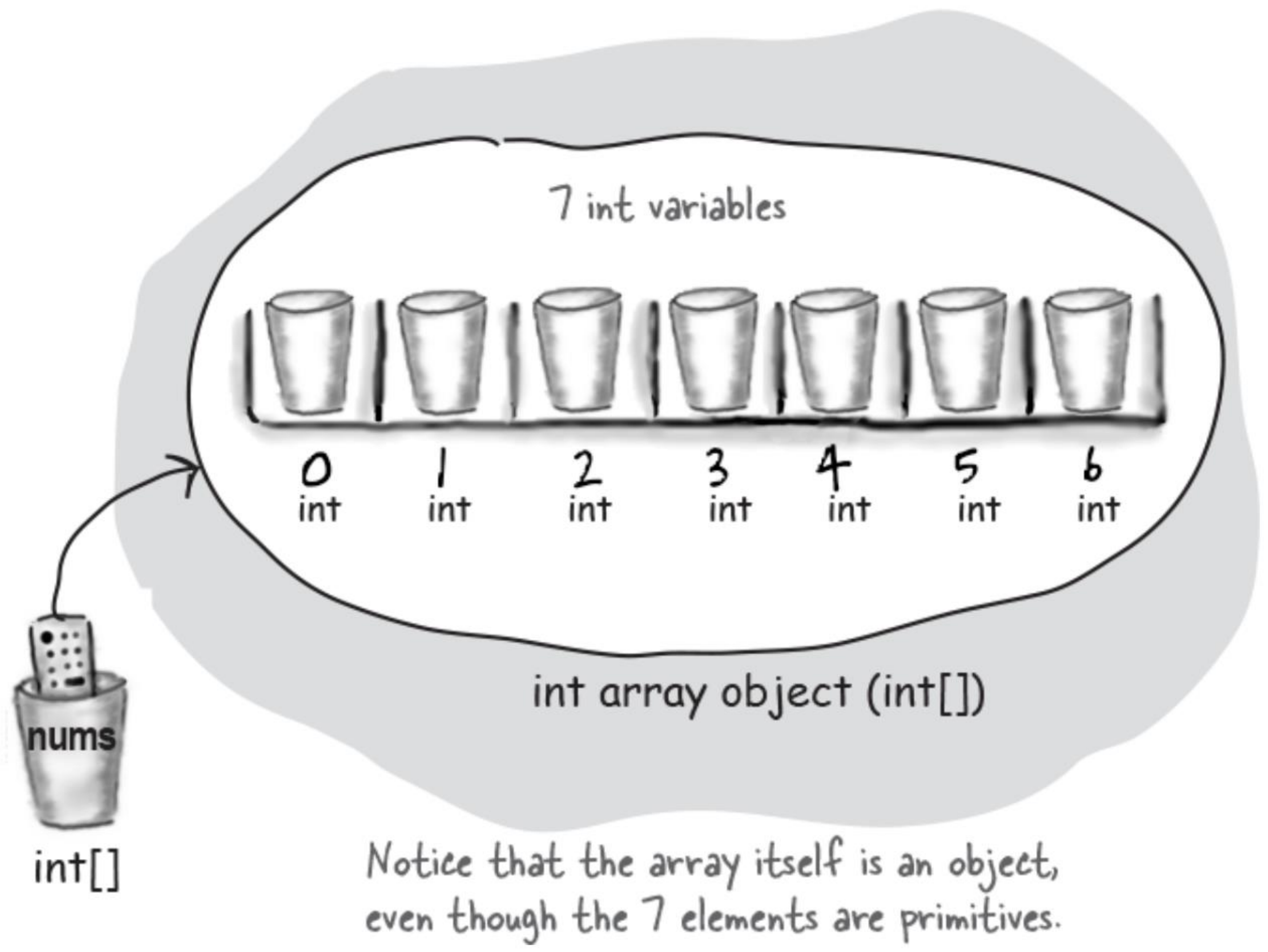
Arrays are always objects, whether they're declared to hold primitives or object references.

```
int[] nums;
```

```
nums = new int[7];
```


7 int variables

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```

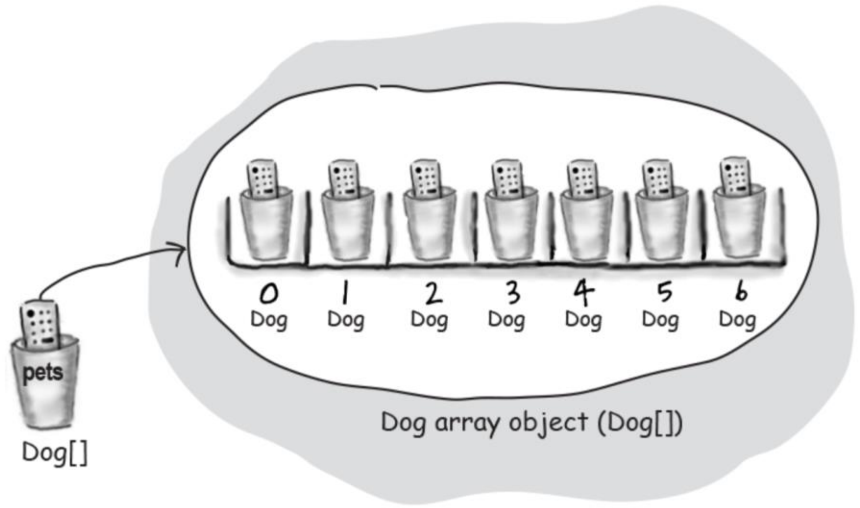

7 int variables
0 int
1 int
2 int
3 int
4 int
5 int
6 int
int array object (int[])
nums
int[]
Notice that the array itself is an object, even though the 7 elements are primitives.

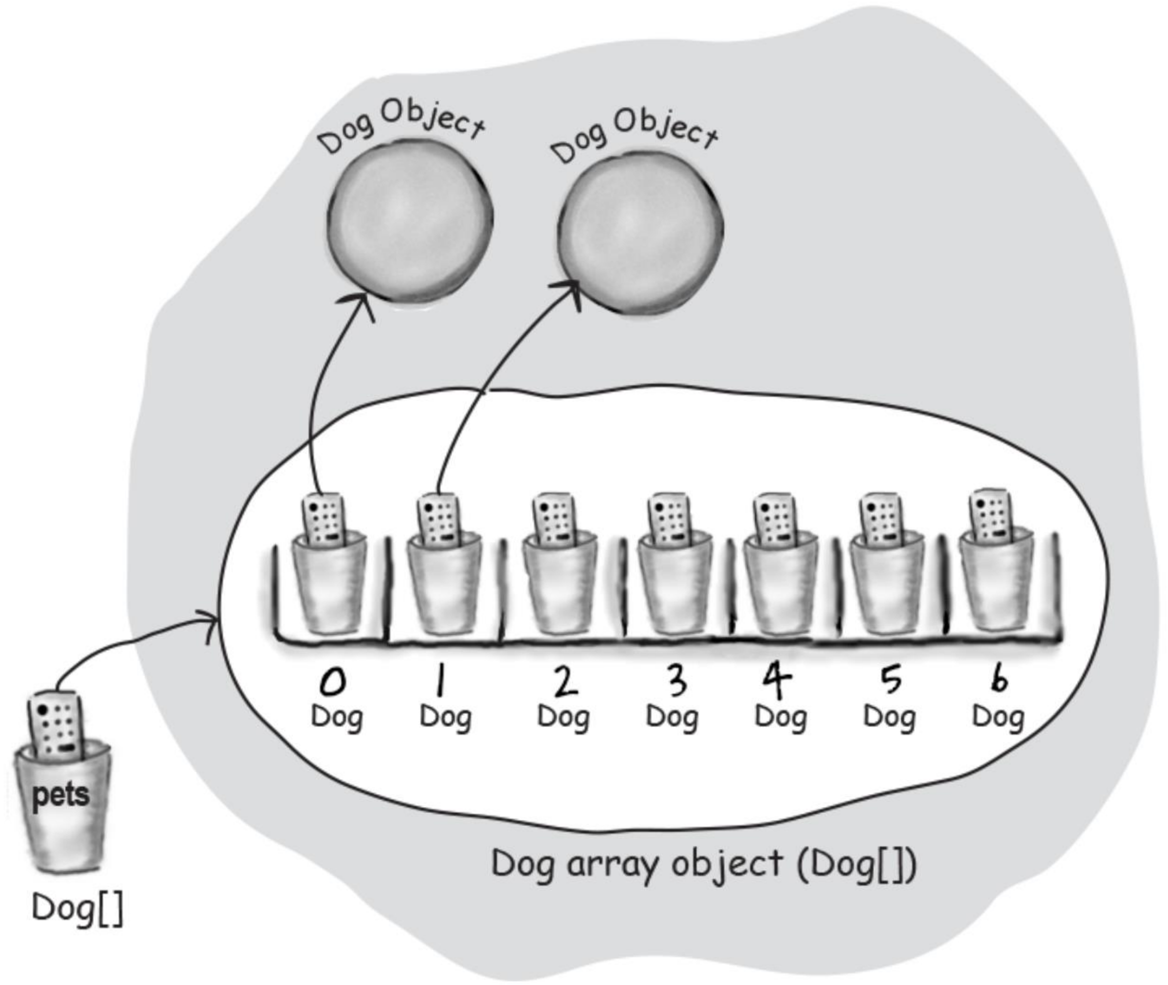
Array

```
pets[0] = new Dog();
pets[1] = new Dog();
```

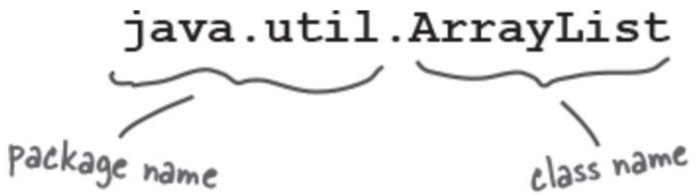
```
Dog[] pets;
```

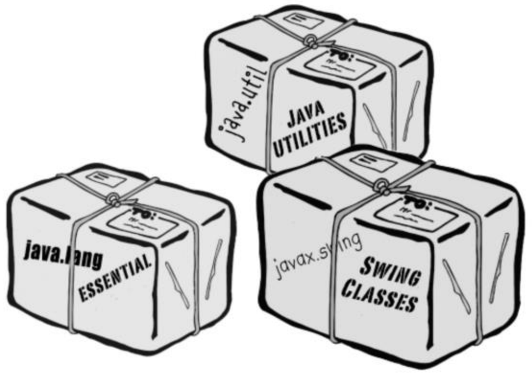
```
pets = new Dog[7];
```

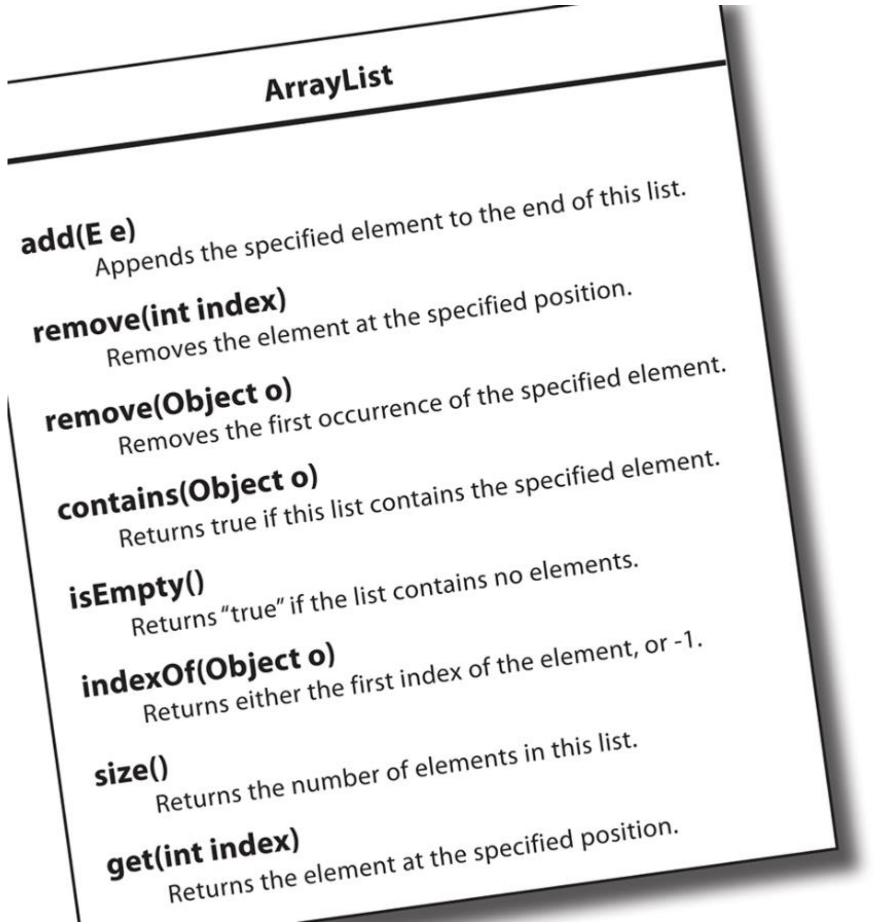




new **ArrayList**<*NoPrimitivesAllowed*>();







ArrayList

add(E e)
Appends the specified element to the end of this list.

remove(int index)
Removes the element at the specified position.

remove(Object o)
Removes the first occurrence of the specified element.

contains(Object o)
Returns true if this list contains the specified element.

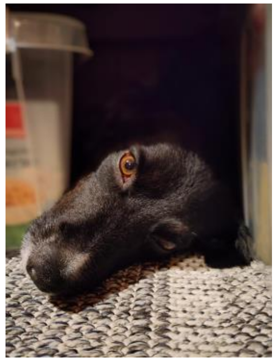
isEmpty()
Returns "true" if the list contains no elements.

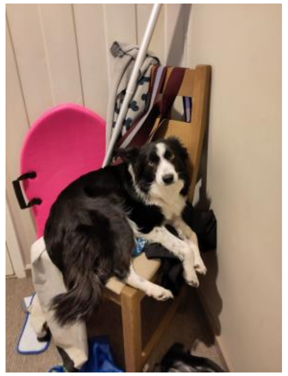
indexOf(Object o)
Returns either the first index of the element, or -1.

size()
Returns the number of elements in this list.

get(int index)
Returns the element at the specified position.

```
import java.util.ArrayList;
public class MyClass {… }

ArrayList<Dog> myDogs = new ArrayList<Dog>();
```





Demo : java is pass-by-value

COLLABORATIVE PROGRAMMING