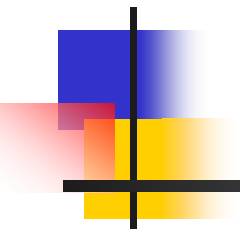
Java Programming



Robert Chatley

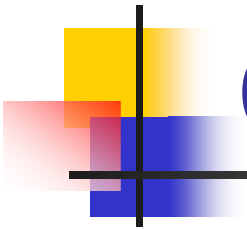
rbc@doc.ic.ac.uk

William Lee

wwhl@doc.ic.ac.uk

Previous Experience

Future *i*



Course Outline

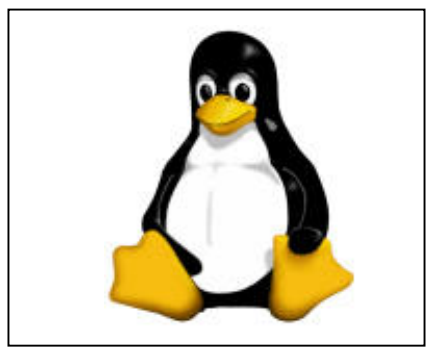
- Day 1 - Introduction
  - Java VM and Memory Model
  - Java Object-Orientation
  - Key language features
- Day 2 - The standard library
  - Collections and ADTs
  - Using containers to store data
  - Generics (new Java 1.5 features)

Course Outline

11 – 12	LEC	LEC
12 – 1	LEC	LEC
1 – 2		
2 – 3	LAB	LAB
3 – 4	LAB	LAB
4 – 5	LEC	LEC

Compiling C++ Programs

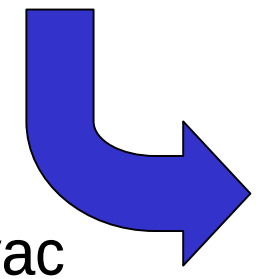
```
#include <stdio>
main( int argc, char *argv[]) {
  // do something
}
```



The Java Virtual Machine

Hello.java

```
class Hello {
  public static void main(String[] args) {
    System.out.println( “Hello World” );
}
```

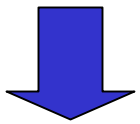


Hello.class

```
…
Method Hello()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return
Method void main(java.lang.String[])
   0 getstatic #2 <Field java.io.PrintStream out>
   3 ldc #3 <String "Hello World!">
   5 invokevirtual #4 <Method void println(String)>
   8 return
```

The Java Virtual Machine

```
class Hello {
  public static void main() {
  // do something
}
```



Hello.class

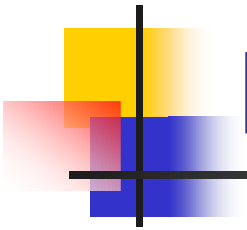
Mac JVM

Hello.class

Linux JVM

Hello.class

Win JVM



Platform Independence

- C++ compiles to native code for a specific architecture (Linux, Windows...)
- Java compiles to Java bytecode
- Same bytecode runs on virtual machine for any platform
  - Only VM is platform specific
  - Good for downloadable code
  - Applets etc

Memory Management (C++)

- C++ has two ways of creating objects

```
int main( int argc , char *[] argv ) {

  Dog d("rover");
  Cat c("fluffy");

  d.bark();

  Dog *dp = new Dog("sirius");

  dp->bark();

  delete dp;
}
```

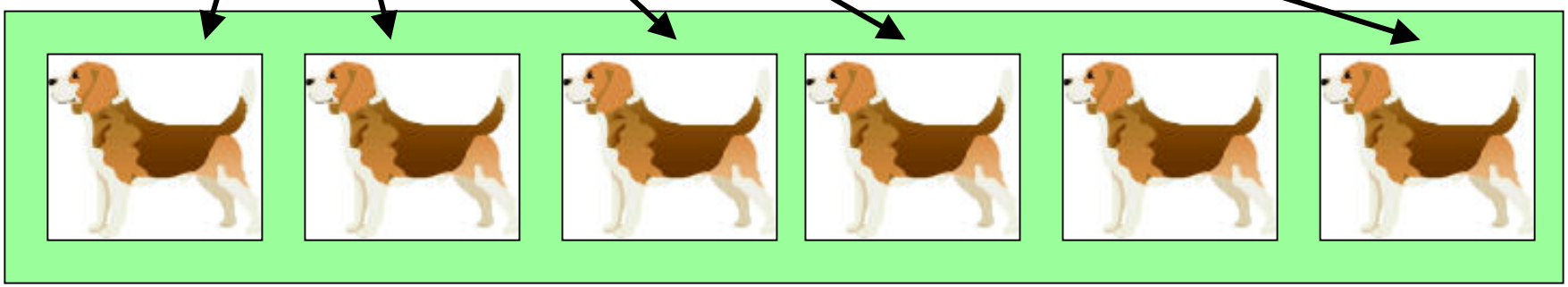
On the stack ("automatic")

On the heap (with new)

Need to delete

Memory Management (C++)

```
int main( int argc , char *[] argv ) {

  for( int i=0; i<100 ; i++ ) {
    Dog *dp = new Dog(“sirius”);
    dp->bark();
  }
}
```



Memory Management (Java)

- All Java objects are created on the heap

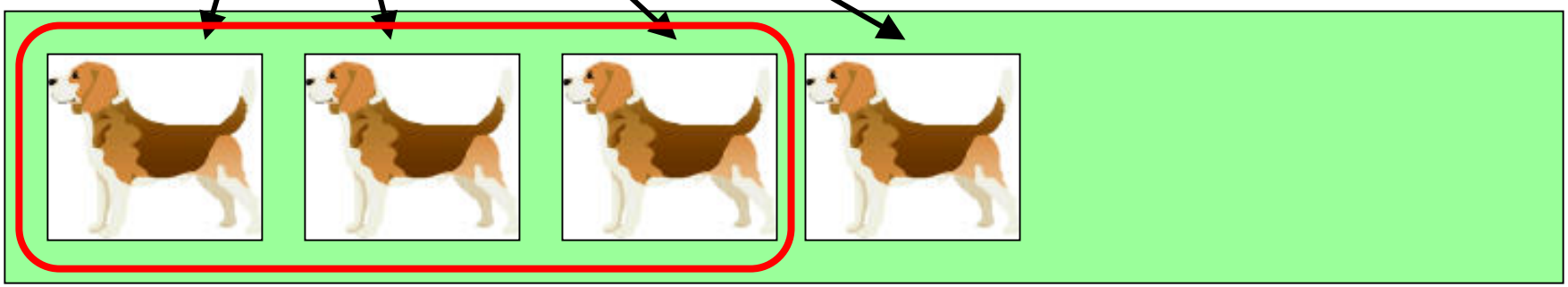
```
public static void main( String[] args ) {

   Dog d = new Dog(“rover”);
   Cat c = new Cat(“fluffy”);

   d.bark();

   Dog dp = new Dog(“sirius”);

   dp.bark();
}
```

No stars or arrows

Memory Management (Java)

```
public static void main( String[] args ){

  for( int i=0; i<100 ; i++ ) {
    Dog dp = new Dog("sirius");
    dp.bark();
  }
}
```

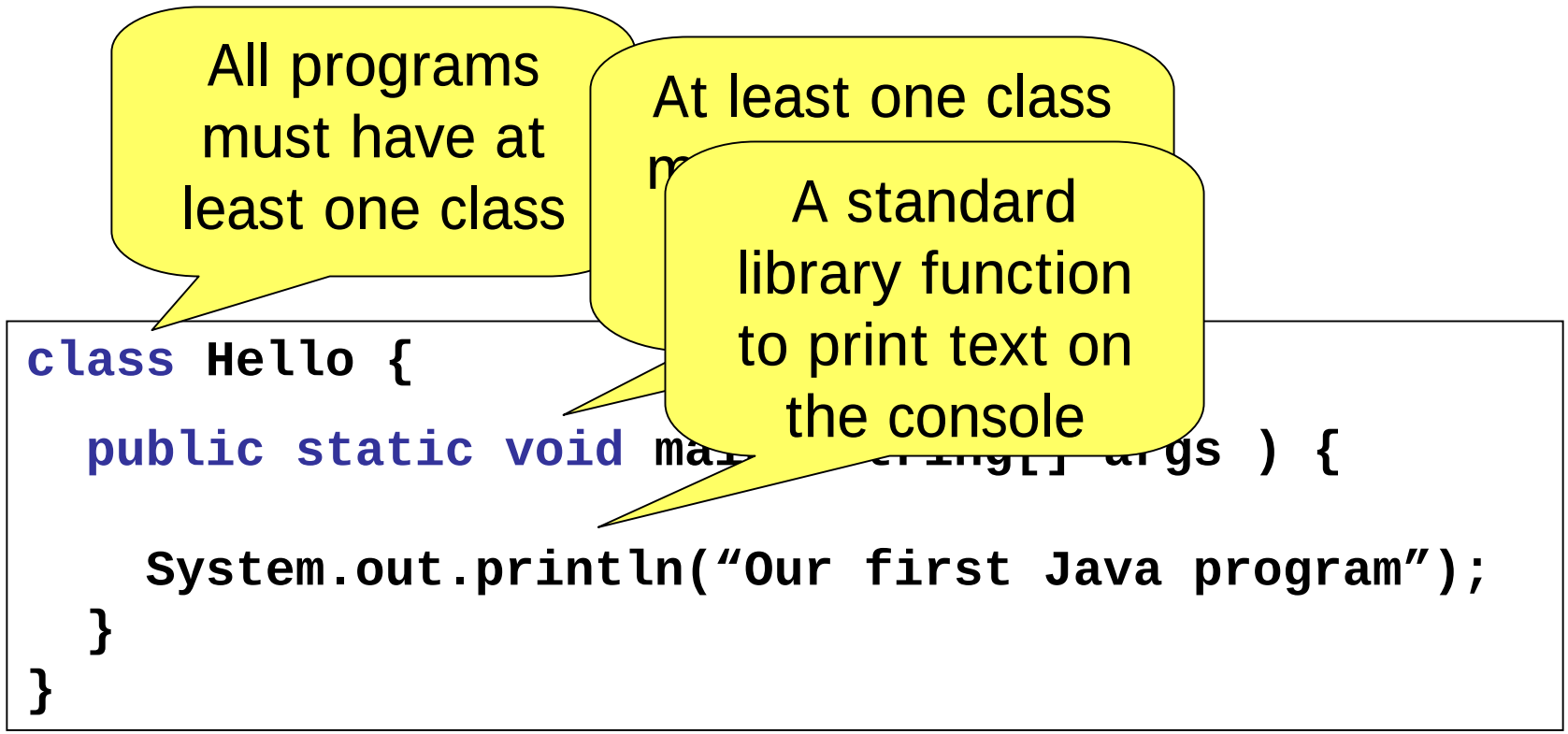
Java's heap is *garbage collected*

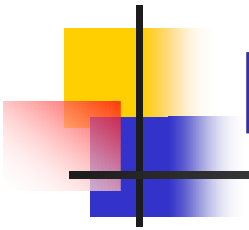


Memory Management (Java)

```
public static void main( String[] args ){

  for( int i=0; i<100 ; i++ ) {
    Dog dp = new Dog(“sirius”);
    dp.bark();
  }
}
```

A First Program

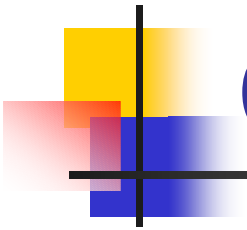




Making it Run

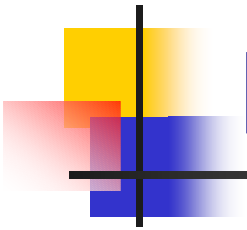
Demo

Text Editor and Command Line Tools



Classpath

- The Classpath tells Java where to look for classes
- Can be an environment variable or flag to java(c)
  - **`setenv CLASSPATH /homes/rbc/classes`**
  - **`java A`**
  - **`java –classpath /homes/rbc/classes A`**
- Often include current dir (.) but can cause problems/confusion



More Classes

- Programs comprise sets of classes
- Classes can have
  - Fields
  - Methods
  - Constructors

A Dog Class

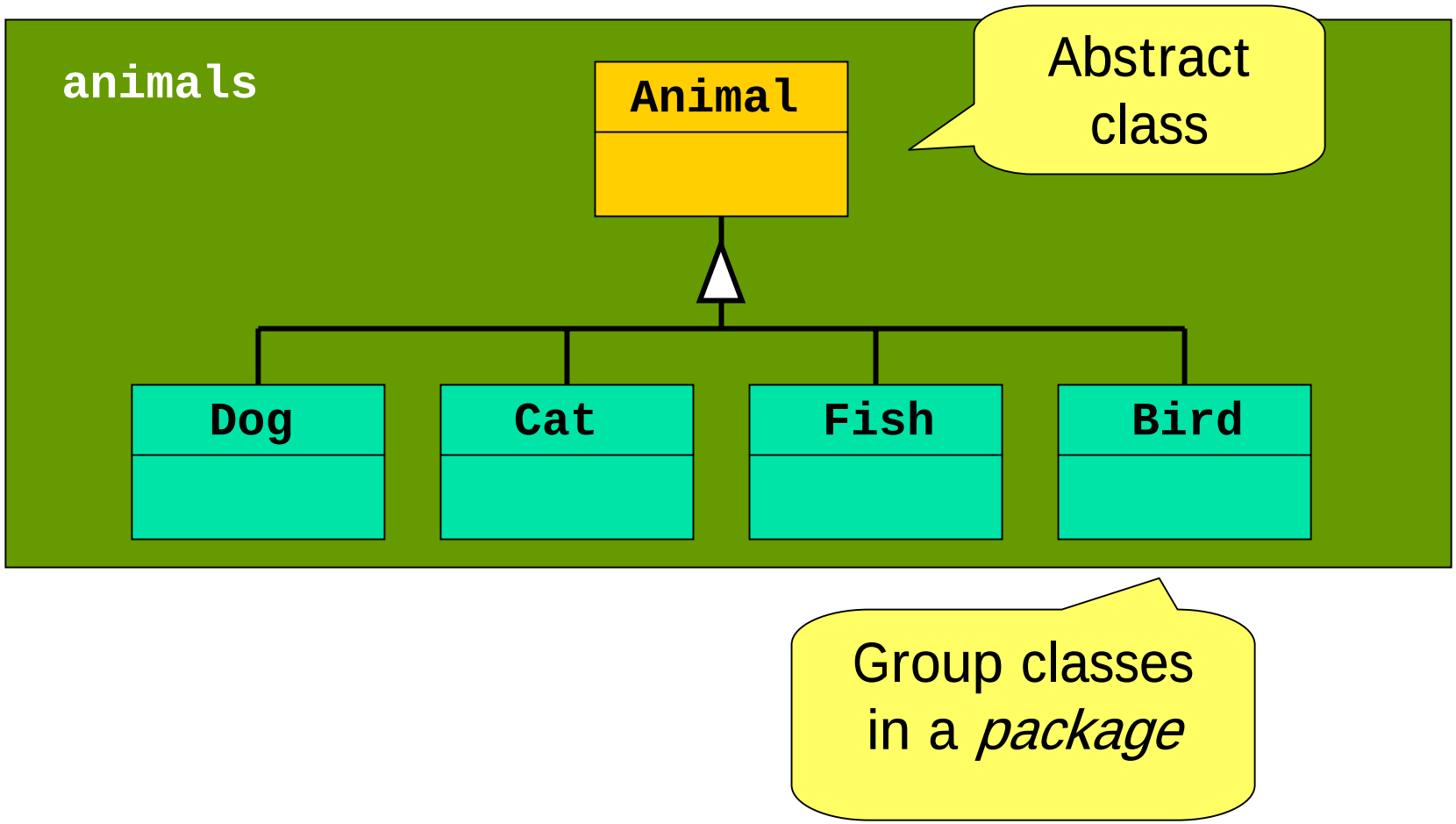
```
class Dog {

  String name;
  int age;

  Dog ( String p_name ) {
    name = p_name;
  }

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

Field declarati

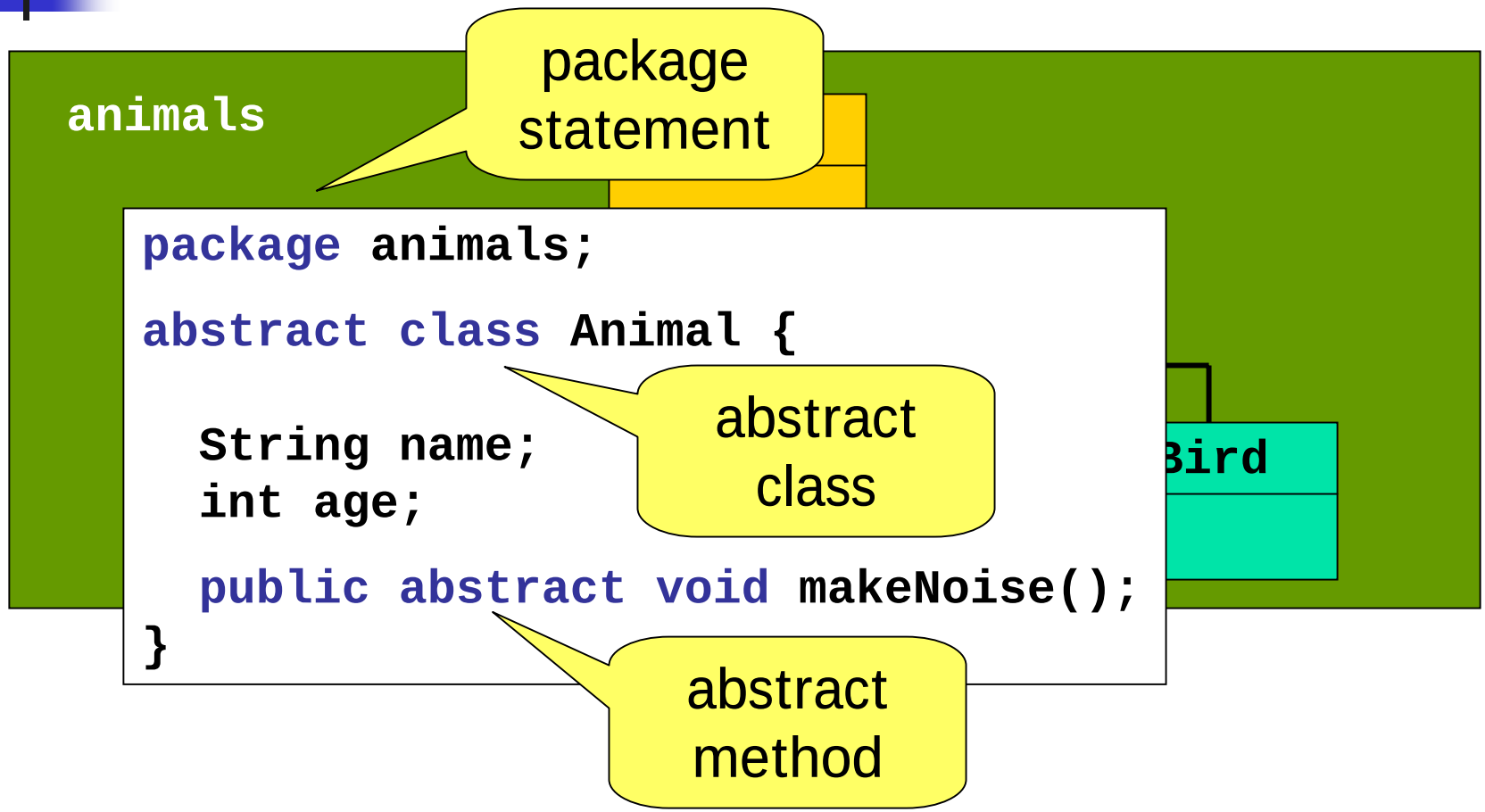
A constructor – NB no destructor as Java has garbage collection

Methods are defined inside the class

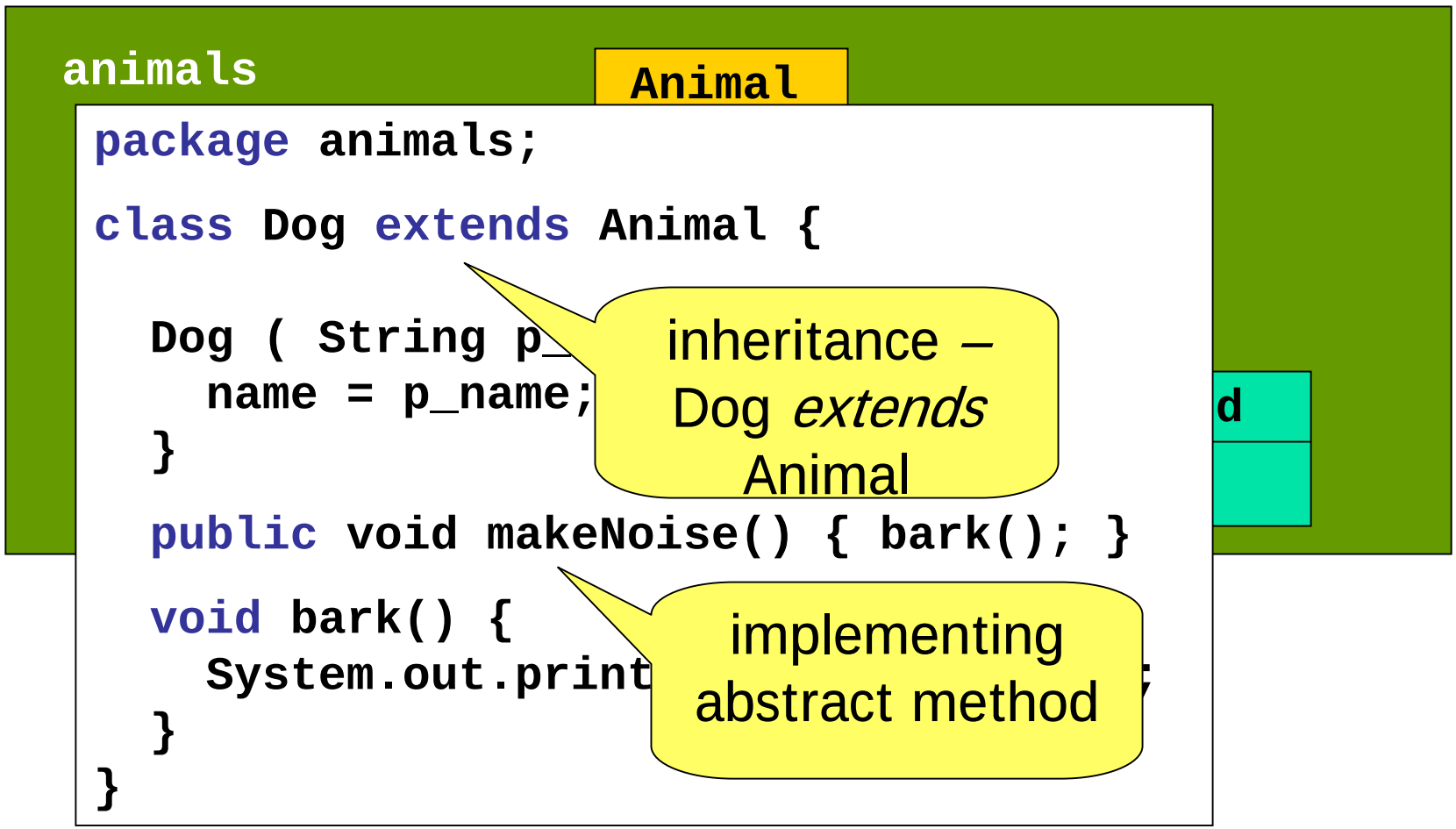
Some other Animals



Some other Animals



Some other Animals



Using our Animals

Import types defined in animals package

```
import animals.*;

class PetShop {

  public static void main( String[] args ) {

    Animal pet = new Dog(“rover”);
    pet.makeNoise();

    pet = new Bird();
    pet.makeNoise();
  }
}
```

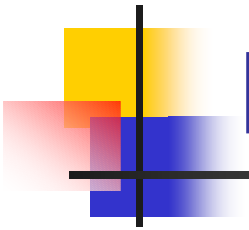
Creating an object with the *new* keyword

Access Modifiers

Classes and methods accessed from outside package must be *public*

```
package a
public class Dog extends Animal {

  public Dog (
    name = p_na
  }

  public voi  m                 (); }

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

Methods called only in this class should be *private*

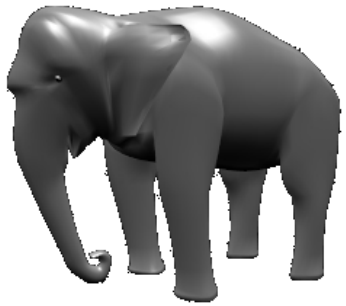


Making it Run

Demo

Java has Single Inheritance

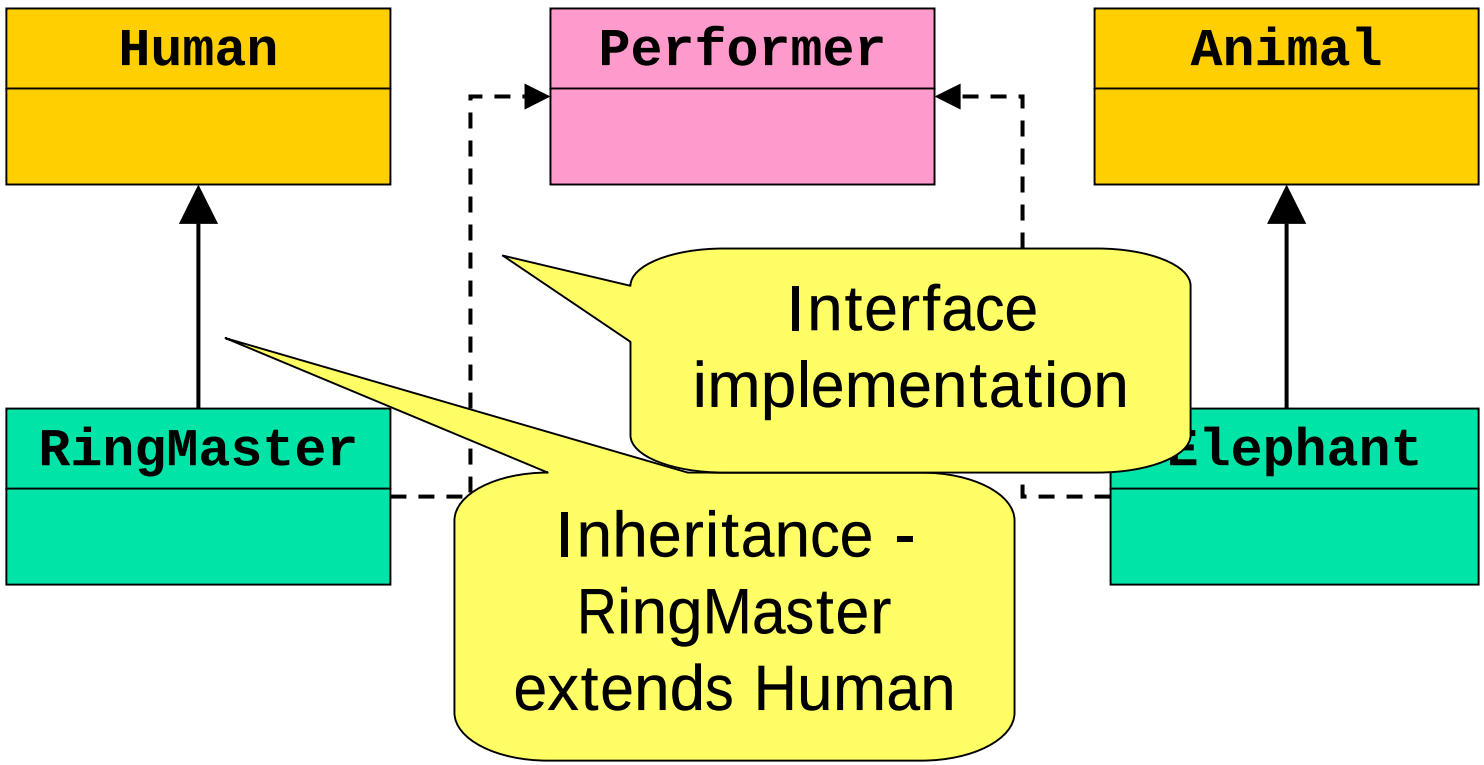
An Elephant is an Animal



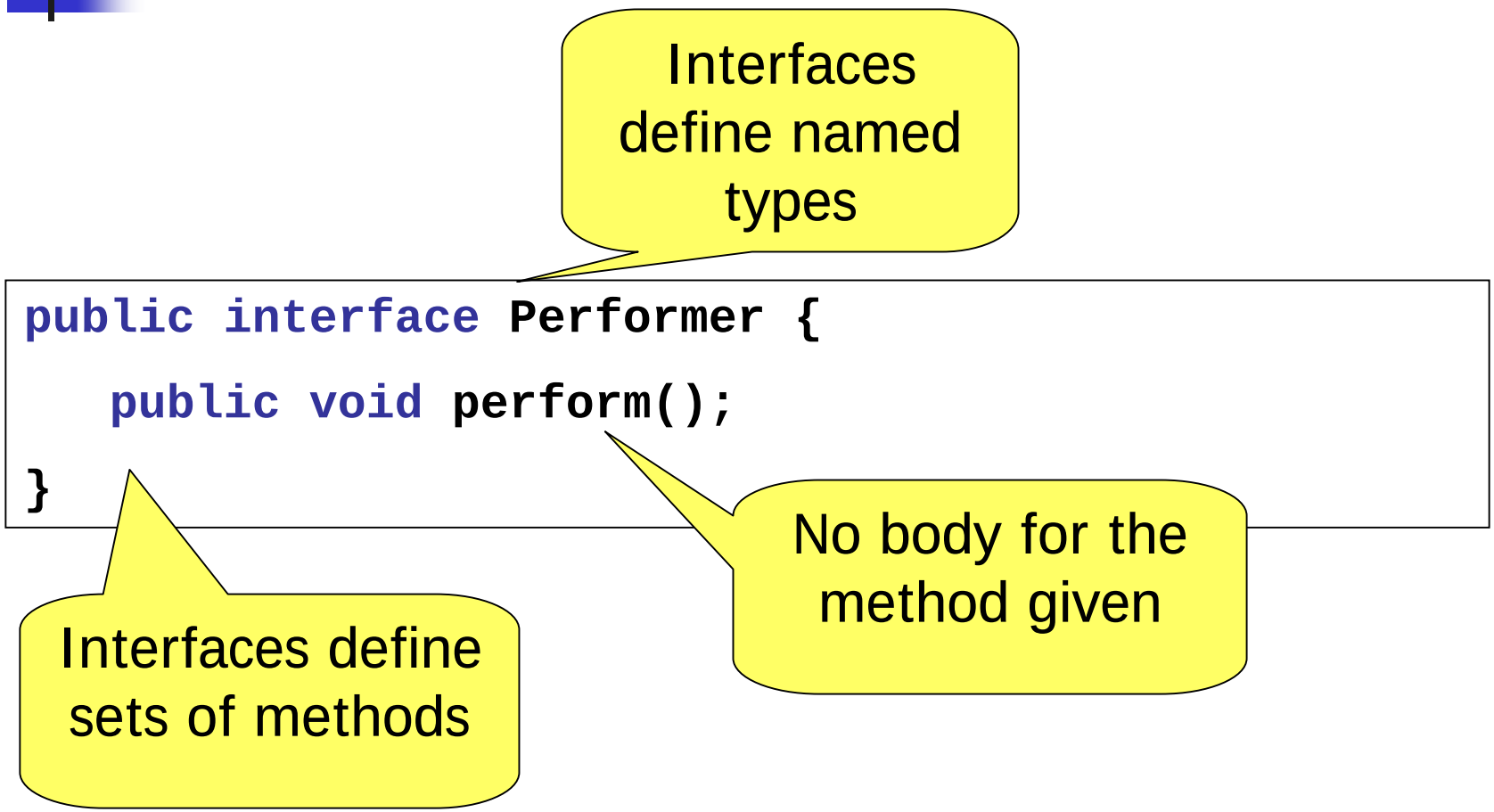
A RingMaster is a Human

What if we want them both to be Performers?

Interfaces


Human
Performer
Animal
RingMaster
Elephant
Interface implementation
Inheritance - RingMaster extends Human

Performer Interface



Implementing the Interface

Declare class as implementing interface

```
public class Elephant extends Animal
                      implements Performer {

   public void makeNoise() { trumpet(); }

   public void perform() { walk(); trumpet(); }

   …
}
```

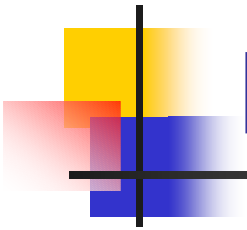
Implement method from Performer

Using Performers

```
import animals.*;

class Circus {

  public static void main( String[] args ) {

    Performer p = new RingMaster();
    p.perform();

    p = new Elephant();
    p.perform();
  }
}
```

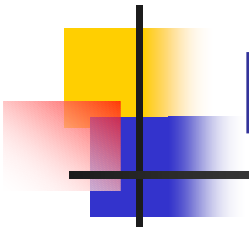
Anything that

Can only call methods from Performer interface on p



Exceptions

- Sometimes a method may fail
  - *Exceptional* behaviour
  - Catch and handle this special case
- Methods can throw an Exception to signal that something is wrong



Documentation for FileReader

FileReader

public **FileReader**(File file) throws FileNotFoundException

Creates a new FileReader, given the File to read from.

Parameters:

file - the File to read from

Throws:

FileNotFoundException - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

Try and Catch

```
class Test {

  …
  public void readFile() {
    File myFile = new File( “data.txt” );
    try {
     FileReader fr = new FileReader( myFile );
    } catch ( FileNotFoundException fnfe ) {
      System.out.println( “File not found.” );
    }
  }
}
```

Things to do in Catch

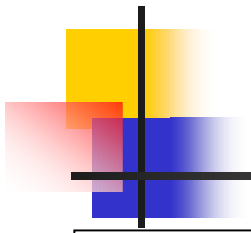
```
class Test {

    …
    } catch ( FileNotFoundException fnfe ) {
      System.err.println( “File not found.” );
      System.err.println( fnfe.getMessage() );
      fnfe.printStackTrace();
    }
  }
}
```

Alternatively, Throw

```
class Test {

  …
  public void readFile() throws
                              FileNotFoundException {
    File myFile = new File( filename );
    FileReader fr = new FileReader( myFile );
  }
}
```

Throwing your own exceptions

```
class Test {

  …

  public void unlockDoor() throws
                              AlreadyOpenException {

  if ( open ) {

    throw new AlreadyOpenException();

  }
}

class AlreadyOpenException extends Exception {}
```



...and Finally

```
public void readFile( filename ) {

  File myFile = new File( filename );
  FileReader fr = new FileReader( myFile );

  try {

    int i = fr.read();

  } catch ( IOException ioe ) {
      System.err.println( “Error reading file” );
  } finally {

    try { if ( fr != null ) fr.close(); }
    catch ( IOException ioe ) {
      System.err.println( “Error closing stream” );
    }
  }
}
```