G51PRG: Introduction to Programming Second semester

Lecture 7

Natasha Alechina
School of Computer Science & IT
`nza@cs.nott.ac.uk`

Previous lecture

- abstract classes
- interfaces
- collections hierarchy in Java

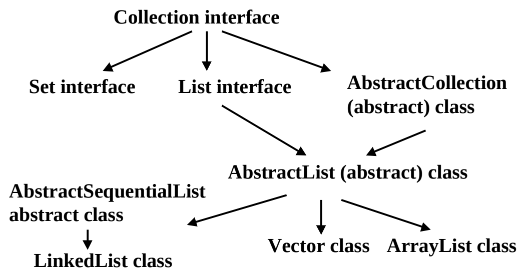
This lecture

- More about interfaces and collections
- test on polymorphism and inheritance
- answer to Book and Textbook exercise
- introduction to the next exercise

How to define an interface

```
public interface Collection{
   public void add(Object o);
   public int size();
…
}
public interface List extends Collection{
   public int indexOf(Object o);
   public Object get(int index);
…
}
```

Example: Java Collections (part of)


Collection interface
Set interface
List interface
AbstractCollection (abstract) class
AbstractList (abstract) class
AbstractSequentialList abstract class
LinkedList class
Vector class
ArrayList class

Interfaces as types

- We can declare a variable of a type corresponding to an interface:

```
List things = new ArrayList();
```

- If a class implements an interface (as Vector implements List) we can use objects of that class when a method requires objects of interface type (as **`sort(List l)`**):

```
Vector myvector = new Vector();
Colections.sort(myvector);
```

(can use **`myvector`** where objects of type AbstractList or List are required, in the usual inheritance polymorphism style).

Why is it useful to implement List

- Some utility methods exist which work for all Collections.
- For example, a method which can sort any data structure of type List.
- Not a separate sorting method for Vectors, a separate method for ArrayLists, a separate method for LinkedLists, but a method for any class implementing List.
- In general utility methods for Collections are held in a class from java.util package (need to import it this package to use Colections!). The class is called Collections.

Java.util.Collections.sort(List list)

- **`public static void sort(List list)`**

This method sorts elements in the list in ascending order using ***natural ordering*** of elements in the list.

- If we are sorting a list of numbers, we know what natural ordering means: the less than relation <.
- What do we do about an arbitrary list of arbitrary things? How do we compare them and decide which one should be before the other?
- In order for the method to work, things in the list must be guaranteed to implement **`compareTo()`** method.
- They way to achieve this in Java is to require that they implement **`Comparable`** interface.

Comparable interface

```
public int compareTo(Object o)
```

- This is the only method in this interface.
- It returns a negative integer if current object is before **`o`** in the natural order, 0 if they are the same, and positive integer if it is after **`o.`**
- Strings implement Comparable (compareTo() supports lexicographic ordering of Strings).
- Numbers implement Comparable (compareTo() supports ordering of numbers).

Example: Integers

```
public int compareTo(Object o){
   return (this.intValue() -
            ((Integer) o).intValue());
}
```

- This is how we could have implemented compareTo() for Integers.
- Note that we need to cast o to Integer.
- Often people return -1 if this object is less than o, 0 if they are the same, 1 if this is greater than o.

Example: sorting a Vector

For any objects which implement Comparable, in this case Integers:

```
Vector myVector = new Vector();
myVector.add(new Integer(5));
myVector.add(new Integer(3));
myVector.add(new Integer(7));
Collections.sort(myVector);
// now myVector is sorted in natural
// order of Integers
```

Example: iterating through a Vector

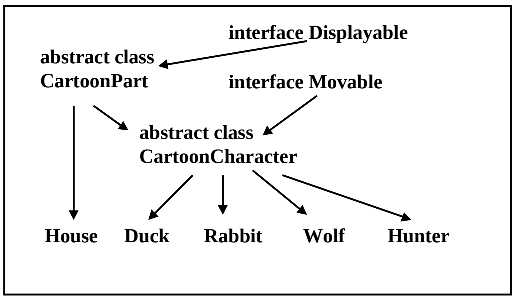
- Here is a simple minded iteration not using an **`Iterator`** object.

```
Vector myVector = new Vector();
myVector.add(new Integer(5));...
for(int i = 0; i < myVector.size(); i++){
  System.out.println(
  ((Integer)myVector.get(i)).intValue());
}
```

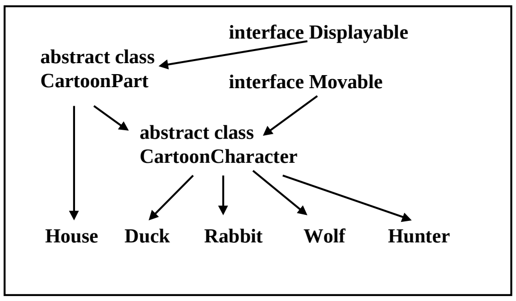
Test

- 5 minutes to look at the classes and interfaces
- it is useful to draw a class hierarchy
- try to answer the questions
- then I’ll go through them and explain

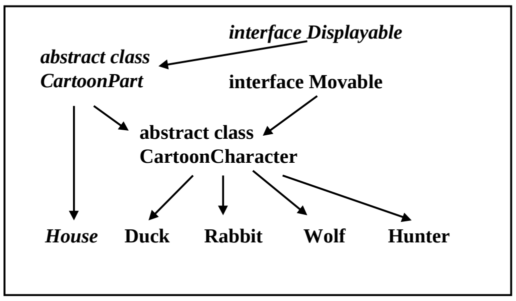
Cartoon Characters


interface Displayable
abstract class CartoonPart
interface Movable
abstract class CartoonCharacter
House
Duck
Rabbit
Wolf
Hunter

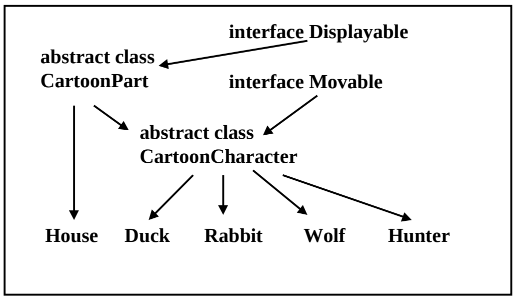
Does House have display() method?


interface Displayable
abstract class CartoonPart
interface Movable
abstract class CartoonCharacter
House
Duck
Rabbit
Wolf
Hunter

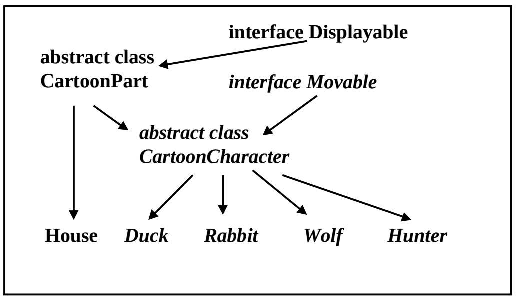
Yes, House has display() method


interface Displayable
abstract class CartoonPart
interface Movable
abstract class CartoonCharacter
House
Duck
Rabbit
Wolf
Hunter

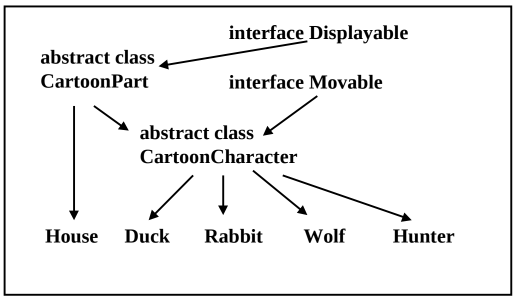
Does House have moveLeft()?


interface Displayable
abstract class CartoonPart
interface Movable
abstract class CartoonCharacter
House
Duck
Rabbit
Wolf
Hunter

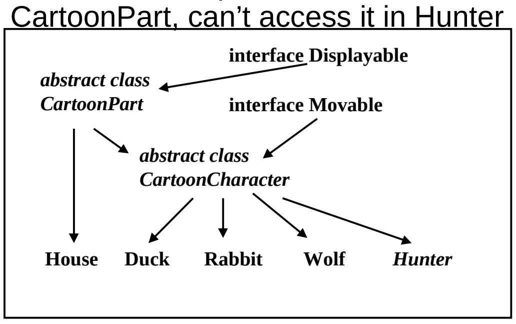
Does House have moveLeft()? No!


interface Displayable
abstract class CartoonPart
interface Movable
abstract class CartoonCharacter
House
Duck
Rabbit
Wolf
Hunter

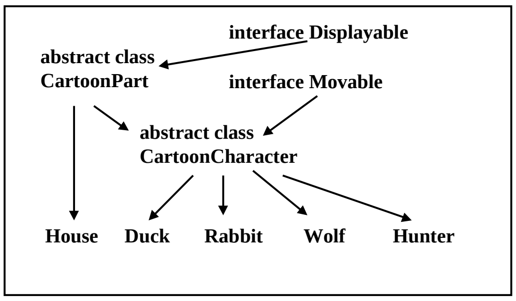
Does Hunter have images field?



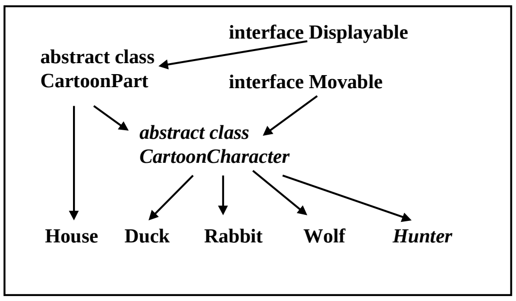
Does Hunter have images field? No: it’s declared as private in CartoonPart, can’t access it in Hunter



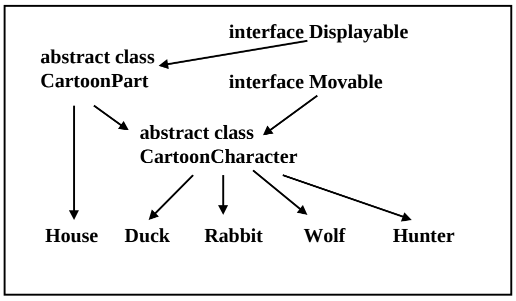
Does Hunter have panic() method?



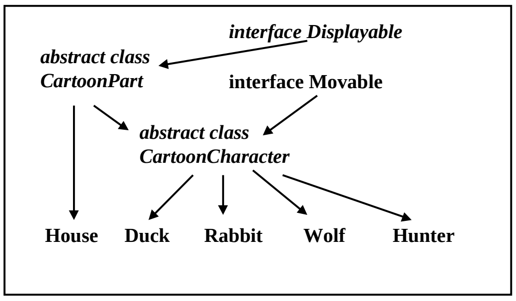
Does Hunter have panic() method? Yes.



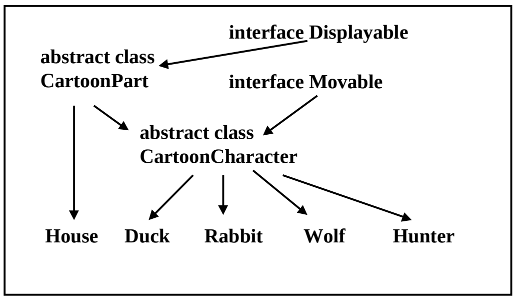
CartoonCharacter for Displayable?



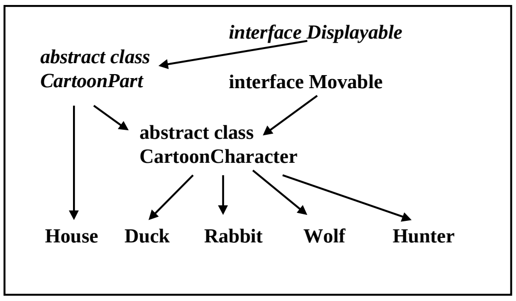
CartoonCharacter for Displayable? Yes!



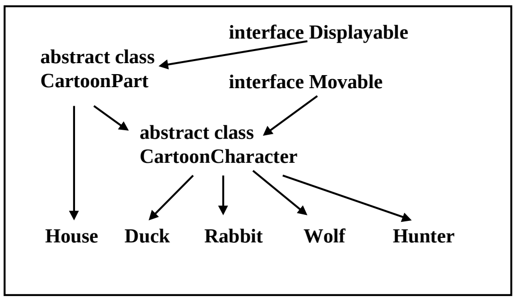
CartoonPart for Displayable?



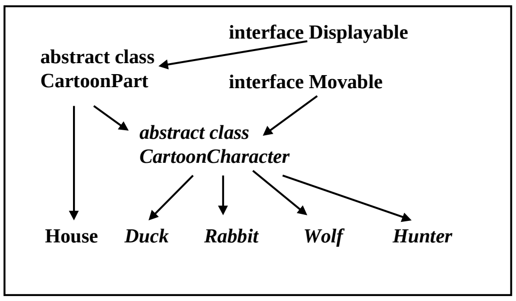
CartoonPart for Displayable? Yes!



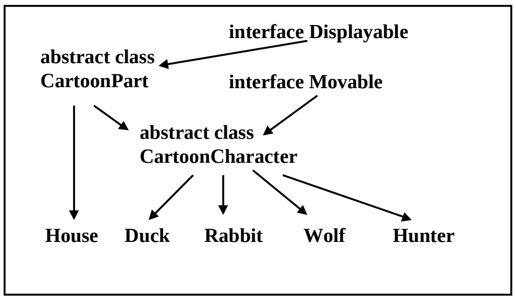
Displayable for CartoonCharacter?



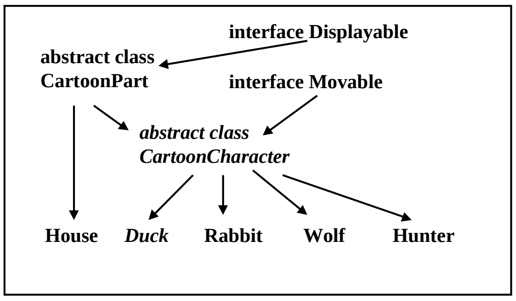
Displayable for CartoonCharacter? No!



Duck for CartoonCharacter?



Duck for CartoonCharacter? Yes!



New exercise

- Implement a class hierarchy

$$\begin{tikzcd}
 & \textbf{Publication} \arrow[dl] \arrow[d] \arrow[dr] & \\
\textbf{Book} & \textbf{Article} & \textbf{TechReport}
\end{tikzcd}$$

New exercise

- Make sure all publications implement Comparable
- Get a list of publications from the user
- Put them in some collection which implements List interface (LinkedList, Vector, ArrayList)
- Sort them in alphabetical order using Collections.sort()

Previous exercise: Book class

- Write a class Book which has fields author, title, publisher, year, registration number.
- Book constructor takes author, title, publisher (Strings), year (int).
- Registration number (int) is generated by counting how many Book objects have been created.
- Books have print() method which prints all the fields.

Book constructor

```
class Book {
 static int count = 1;
 private String author, title, publisher;
 private int year, number;
 public Book(String a, String t, String
  p, int y) {
  this.author = new String(a);
  this.title = new String(t);
  this.publisher = new String(p);
  this.year = y;
  this.number = count++;
 }
```

Book print()

```
 public void print() {
  System.out.println(author);
  System.out.println(title);
  System.out.println(publisher + ", " +
  year);
  System.out.println("Library number " +
  number);
 } // end print
} // end class Book
```

Previous exercise: Textbook

- Write a class Textbook which extends Book.
- Has additional field String course.
- Constructor takes author, title, publisher (Strings), year (int), course (String).
- Registration number is generated by counting how many Book or Textbook objects have been created.
- print() method prints all the fields + course.

Textbook

```
class Textbook extends Book {
 private String course;
 public Textbook(String a, String t,
  String p, int y, String c) {
  super(a, t, p, y);
  this.course = new String(c);
 } // end constructor
 public void print() {
  super.print();
  System.out.println("Course " + course);
 } // end print
} // end class Textbook
```

Summary and further reading

- Abstract classes and interfaces allow Java programmers to implement methods at the right place in the class hierarchy and re-use code.
- I covered general principles of extending classes, implementing interfaces, and using methods polymorphically, but only a tip of the iceberg in Collection classes and other library methods.
- If you are interested look at Iterators and Comparators.
- For class hierarchies and interfaces, read

http://java.sun.com/docs/books/tutorial/java/javaOO/subclasses.html
http://java.sun.com/docs/books/tutorial/java/interpack/interfaces.html