LECTURE 6: POLYMORPHISM

CS 2110
Fall 2021

Agenda

Previously in 2110:

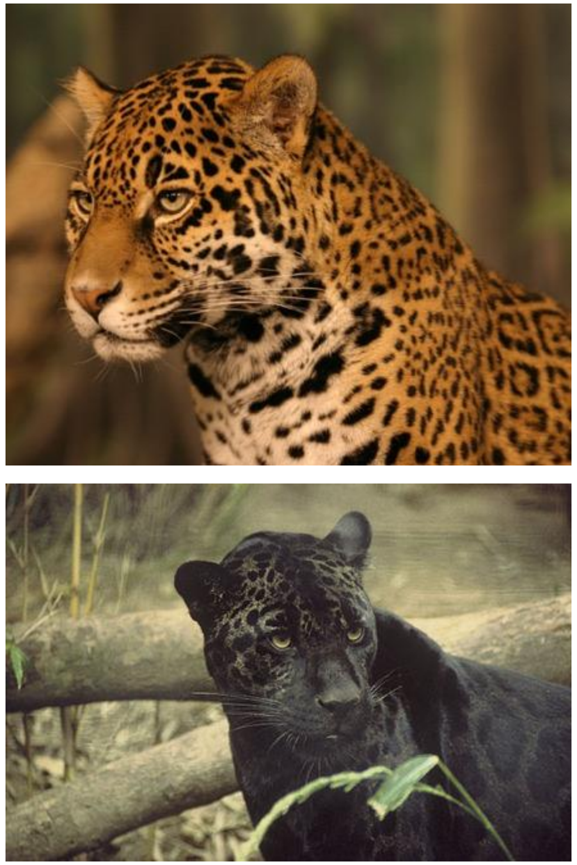
- Objects and classes
- Encapsulation
- Inheritance: subclassing and overriding

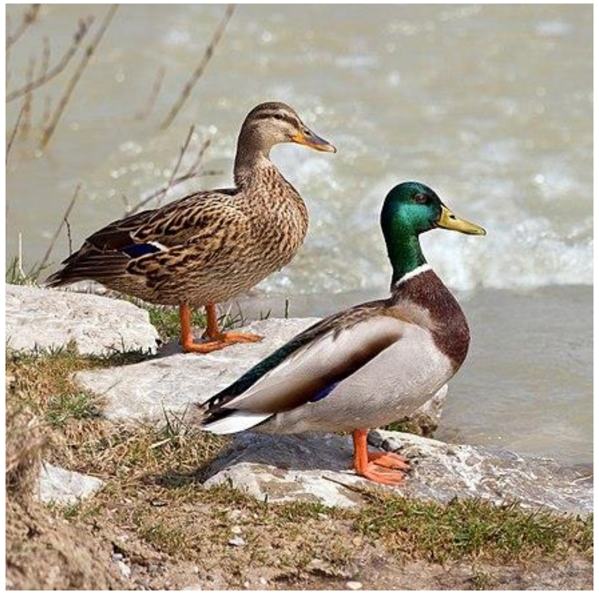
Today:

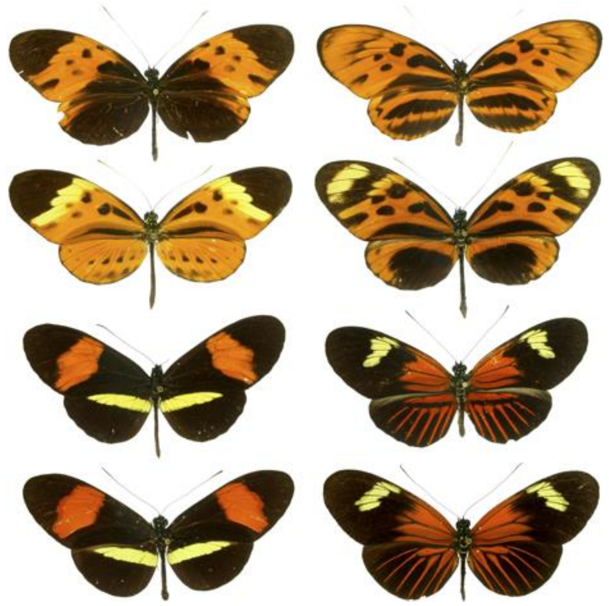
- Polymorphism
- Overloading
- Wrapper classes, autoboxing, and coercion
- Parameterized classes
- Subtyping

Polymorphism

Gk. poly = many, morph = form


Jaguar: light and dark morphs


Mallard: sexual dimorphism


Heliconius butterflies

Same yet different

Images: https://en.wikipedia.org/wiki/Polymorphism_(biology), https://en.wikipedia.org/wiki/Polymorphism_in_Lepidoptera

Polymorphism in Programming

- Polymorphism: language treats as though same, despite differences
- General phenomenon with three manifestations in Java:
  - **Ad-hoc** polymorphism
  - **Parametric** polymorphism
  - **Subtype** polymorphism

Ad-hoc Polymorphism

Recall: Overloading

```
class Counter {
    int count;
    …
    void reset() {
        count= 0;
    }
    void reset(int i) {
        count= i;
    }
}
```

Overloaded method:
Same name, different signatures.
Polymorphic message.

Ad-hoc Polymorphism

- Ad-hoc polymorphism: A method/operator appears to be applicable to several different types
  - Those types need not share a common structure
  - Method/operator may behave in unrelated ways for each type
- Examples in Java:
  - Overloading of methods
  - Overloading of +
  - Coercion
  - Autoboxing…=

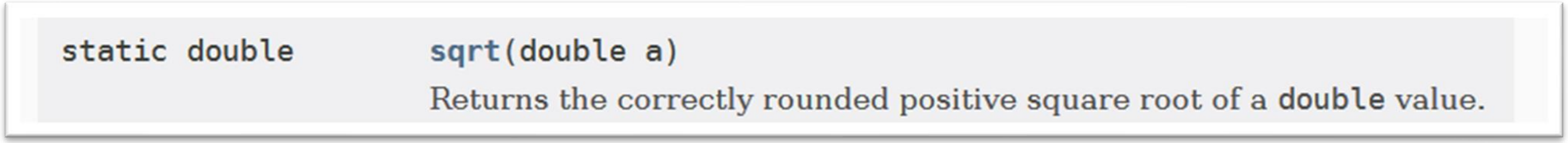
Type conversions

- **Casting**
  (explicit conversion)

- `// Integer division`
  `5 / 2 ==> 2`
- `// Floating-point division`
  `(double) 5 / (double) 2 ==> 2.5`

- **Coercion**
  (implicit conversion)

-
  ```
  Math.sqrt(4) ==> Math.sqrt(4.0)
               ==> 2.0
  ```



`static double` `sqrt(double a)`

Returns the correctly rounded positive square root of a `double` value.

Java Wrapper Classes

- Wrapper class: wrap a value of a primitive type inside an object
- Primitive types: built-in; not defined by a class

Primitive type	Wrapper class
`int`	`Integer`
`char`	`Character`
`boolean, byte, short, long, float, double`	`Boolean, Byte, Short, Long, Float, Double`

LECTURE 6: POLYMORPHISM

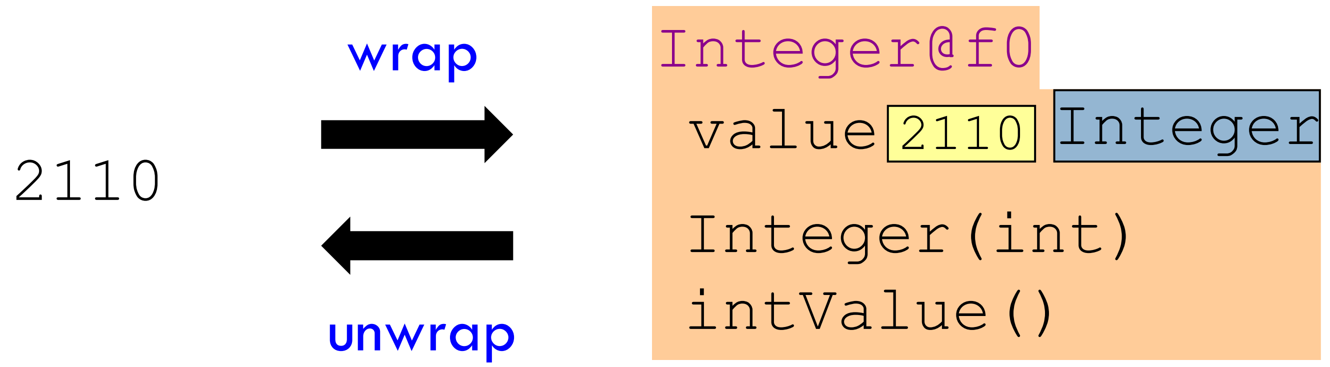
PART 2: WRAPPER CLASSES AND AUTOBOXING

CS 2110
Fall 2021

Wrapping and Unwrapping

Java automatically coerces between

- value of primitive type, and
- object of wrapper class



Java calls this feature autoboxing

Ad-hoc Polymorphism and Autoboxing

Recall: Ad-hoc polymorphism: A method/operator appears to be applicable to several different types

```
void printInt(int i) { System.out.println(i); }
printInt(new Integer(2110));
```

really:

```
printInt(new Integer(2110).intValue());
```

Ad-hoc Polymorphism

Why is it polymorphic?

- Overloading: Same method/operator has “many forms”
- Autoboxing: Same value has “two forms”

LECTURE 6: POLYMORPHISM

PART 3: PARAMETRIC POLYMORPHISM AKA «GENERICS»

CS 2110
Fall 2021

Specialized Boxes

```
public class IntBox {
    private int contents;
    public void put(int t) {
        contents= t;
    }
    public int get() {
        return contents;
    }
}
```

```
public class DoubleBox {
    private double contents;
    public void put(double t) {
        contents= t;
    }
    public double get() {
        return contents;
    }
}
```

Too much code duplication!

Polymorphic Boxes

```
public class Box<T> {
    private T contents;
    public void put(T t) {
        contents= t;
    }
    public T get() {
        return contents;
    }
}
```

Demo

Building a box

Polymorphic Boxes

Constructor:

```
public Box(T t) { put(t); }
```

New expression:

```
Box<Integer> b= new Box<Integer>(new Integer(1))
Box<Integer> b= new Box<Integer>(1)
Box<Integer> b= new Box<>(1)
```

Parametric Polymorphism

Why is it polymorphic?

- Class is parameterized on a class type
- Parameter is instantiated when object is created
- So same class has “many forms”
- aka “generic” in type parameter

LECTURE 6: POLYMORPHISM

PART 4: SUBTYPE POLYMORPHISM

CS 2110
Fall 2021

Recall: Bank Accounts

```
class Account { …
    void printStatement() { … } }

class InterestAccount extends Account { …
    @Override void printStatement() { … } }
```

How could a bank print statements for all its accounts?

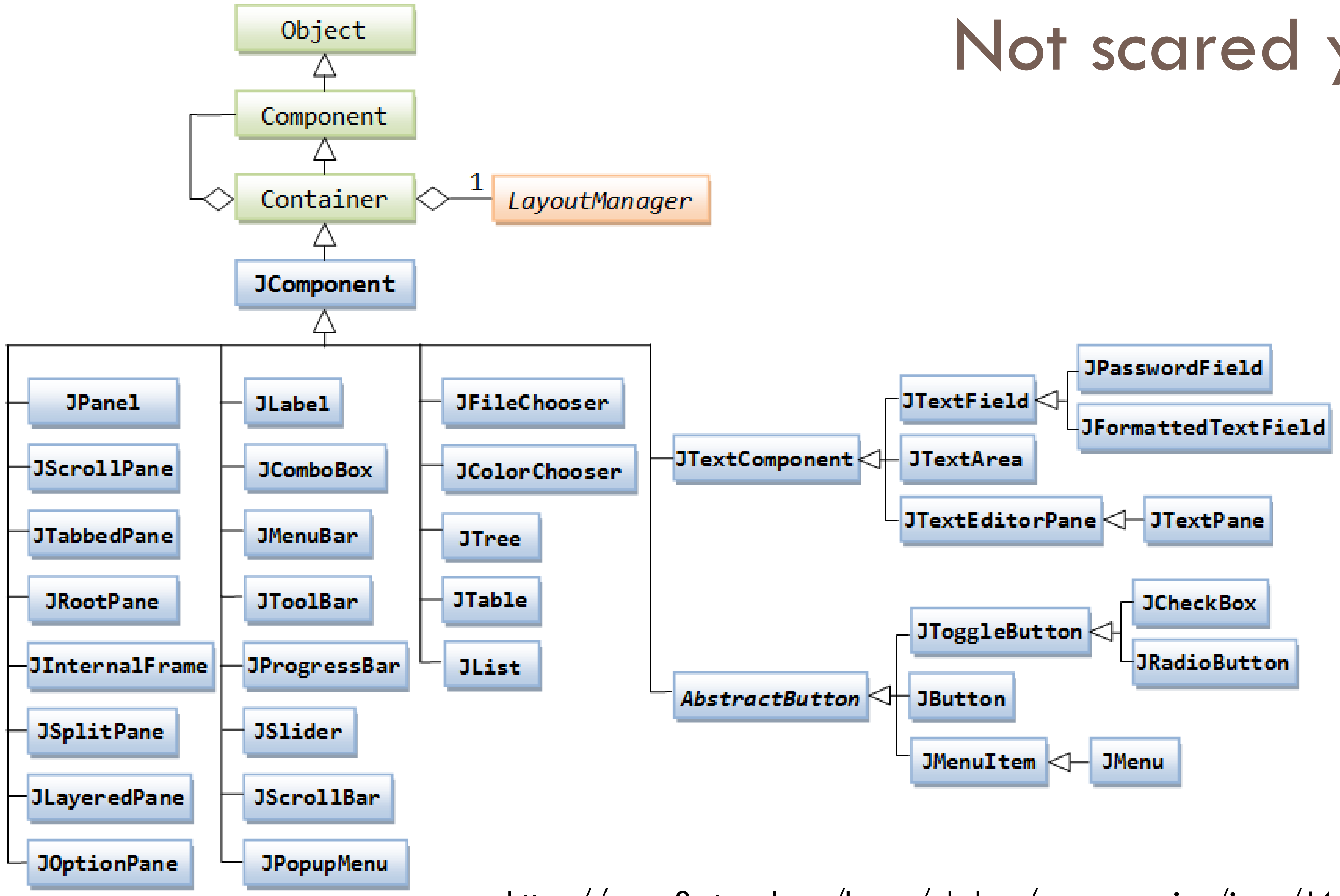
Printing All Statements

```
Account[] accounts= …;
InterestAccount[] interestAccounts= …;

for (Account a: accounts) {
    a.printStatement();
}
for (InterestAccount a: interestAccounts) {
    a.printStatement();
}
```

Have to duplicate this code for every kind of account…

Not scared yet?



https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI_2.html

Demo

Subtyping

To be continued…

Your Turn: Read in JavaHyperText

- Overload
- Wrapper class, autoboxing
- Generics, type parameter