Please note that the slides published AFTER the lectures and workshops are the official slides and are the ones that should be used for revision.

COMP2013

Software Maintenance

Lectur

OO and Java Refresher (2/2)

Peer-Olaf Siebers

University of Nottingham
UK | CHINA | MALAYSIA

Week 2 Organisation

- Lecture 2:
  - Going through more advanced Java topics
  - Java Collections framework
  - Implementation of object
- Lab 2:
  - Working further on the ZooApp example
  - Looking at packages
- Workshop 2:
  - CW1 Release
  - IDEs + Java 9/10/11 additions
  - Maintaining the ZooApp (basic maintenance)

java collections framework

Java Collections Framework

- What do we understand by "Collections" in Java?
  - A collection is an object that represents a group of objects
  - The Collections API is a unified framework for representing and manipulating collections, independent of their imple

- What does the abbreviation API stand for
  - Application Programming Interface

- What is the difference between a library and an API?
  - A library contains re-usable chunks of code. These re-usable chunks of code are linked to your program through APIs.

Java Collections Framework

- Java Collections Framework principle ideas:
  - We have container objects that contain objects
  - All containers are either "collections" or "maps"
  - All containers provide a co s, in addition to their unique set of signatures

- The framework contains data structures
  - e.g. arrays; lists; maps

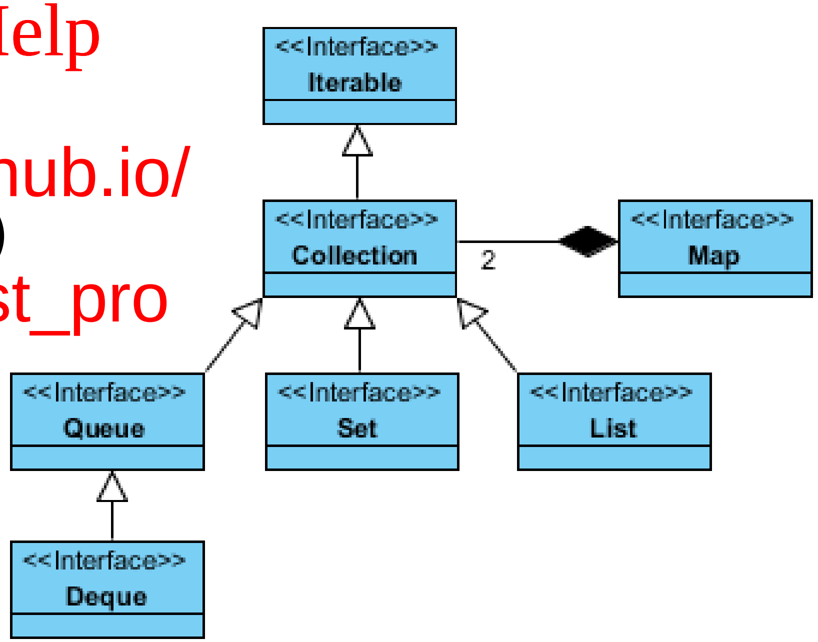
- The framework contains algorithmic operations
  - e.g. searching; sorting

Java Collections Framework

- Collection
  - Something that holds a dynamic collection of objects

- Map
  - Defines mapping between keys and objects (t)

- Iterable
  - Collections are able to return an iterator object that can scan over the contents of a collection one object at a time



Java Collections Framework

- Core collection framework interfaces
  - Iterable: Represents an iterator object
  - Collection: Represents a group of objects (elements)
  - Map: Maps keys to values;
  - Queue: Represents FIFO q
  - Deque: Represents a double ended queue
  - Set: A collection that cannot contain duplicat
  - List: An ordered sequence of elements that allows duplicate elements

- Interface location
  - Most interfaces can be found in the java.util.* package
  - The "Iterable" interface can be found in the java.lang.* package

Java Collections Framework

</>

- Classes that implement the collection interfaces typically have names in the form of <Implementation style><Interface>

- Legacy classes (do not use)
  - Vector (now ArrayList); HashTable (now HashMap); Stack (now ArrayDeque)

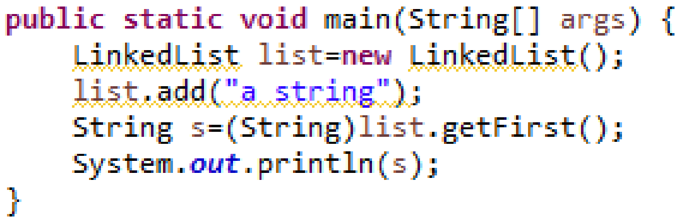
</>

</>

"? extends E" means "some type that either is E or a subtype of E"

Java Collections Framework

- Non typesafe collections (do not use)
  - Collection constructors are not able to specify the type of objects the collection is intended to contain
  - Need to cast objects when on" will be thrown if we attempt to cast to the wrong type



```
public static void main(String[] args) {
    LinkedList list=new LinkedList();
    list.add("a string");
    String s=(String)list.getFirst();
    System.out.println(s);
}
```

Java Collections Framework

- Typesafe collections with "Generics"
  - Classes support generics by allowing a type variable to be included in their declaration; type are declared for the reference and constructor

- You cannot type a collection using a primitive type
  - Values of primitive types need to be put into objects of a suitable wrapper class before they can be added to a collection

ArrayList Class

TreeSet Class

- TreeSet provides an implementation of the Set interface that uses a tree for storage. Objects are stored in sorted, ascending order.

HashMap Class

- HashMap is a Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key

Java Collections Examples

</>

implementati riented principles

Aggregation and Composition; Inheritance; Polymorphism; Abstract Methods and Classes; Interfaces

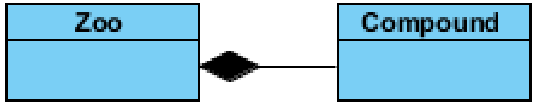
Case Study: Zoo Management

</>

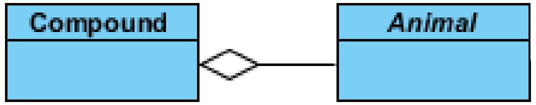
Aggregation and Composition

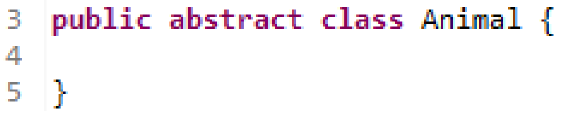
- What is the difference between the Aggregations and Compositions?
  - Aggregation
    - The object exists outside is passed as an argument (for example) to the constructor

  - Composition
    - The object only exists, or only makes sense inside the other, as a part of the other



Aggregation

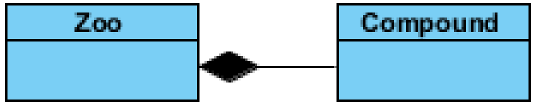




```java
public abstract class Animal {

}
```

Composition



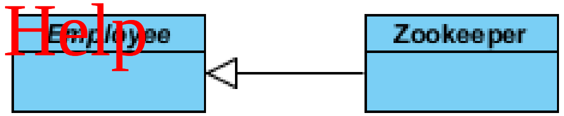
Inheritance

- What is inheritance and why do we use it?
  - Inheritance: Forming new classes based on existing ones
    - A way to share/reuse co
  - Superclass: Parent class being extended
  - Subclass: Child class that inherits behavior fro
    - Gets a copy of every field and method from superclass
  - "is-a" relationship: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one

Inheritance

- Example:

```
public class Zookeeper extends Employee {
...
}
```



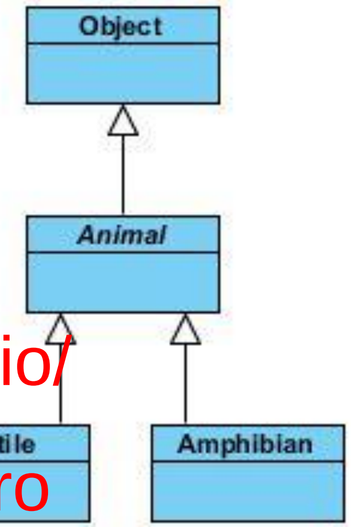
- By extending Employee, each Zookeeper o
  - Receives a copy of each method from Employ lly
  - Can be treated as an Employee by client code
  - Zookeeper can replace ("override") behavior from Employee

Inheritance

- A subclass can call its parent's method/constructor:

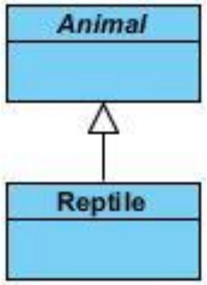
Inheritance

- Every class is either
  - a direct subclass of Object (no extends)
  - a subclass of a descendant of Object (extends)



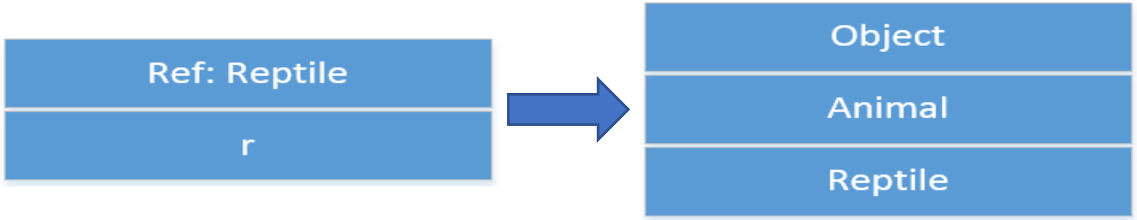
- Class Reptile extends Animal
- Class Amphibia extends Animal
- Class Animal extends Object

Inheritance



Inheritance

- Object creation process: Reptile r = new Reptile();
  1. Create reference "r"
  2. Start creating Reptile by entering Reptile constructor and making call to parent
  3. Start creating Animal by d making call to parent
  4. Create Object portion
  5. Create Animal portion
  6. Create Reptile portion



Inheritance

- Which of these works?
  - Reptile r = new Reptile();
  - Animal a = new Reptile();
  - Object o = new Reptile();
  - Reptile r = new Animal();
  - Animal a = new Object()

Inheritance

</>

- Casting primitives

```
double d;
float f;
d = f;     // legal...no loss o
f = d;     // illegal...potenti
```

- Casting references

```
Object o;
Reptile r;
o = r;     // legal...a reptile is an object
r = o;     // illegal...not all objects are reptiles
```

Polymorphism

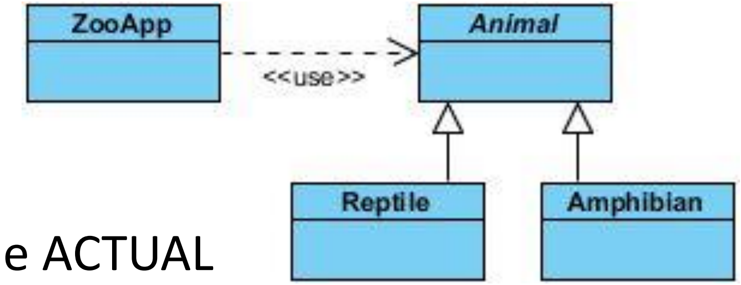
- What is the difference between polymorphism, method overloading, and method overriding?

  - Polymorphism
    - Polymorphism is an obje

    - Method overloading and method overriding a polymorphism

  - Method overloading
    - Methods with same the name co-exists in the same class but they must have different method signature
    - Resolved during compile time (static binding)
  - Method overriding
    - Method with the same name is declared in super and sub class
    - Resolved during runtime (dynamic binding)

Polymorphism



- Dynamic Binding
  - At run time (dynamic) when a method is invoked on a reference the ACTUAL OBJECT is examined and the "lowest" or closest version of the method is actually run.

Abstract Methods and Classes

- Any subclass of class Animal has two choices:
  - Define a eat method (i.e. { })
  - Be abstract

- Note:
  - Abstract classes may not be used to instantiate or make objects (new)
  - References to abstract classes are legal

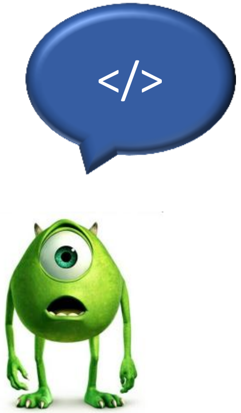
Abstract Methods and Classes

Abstract Methods and Classes

</>

- Abstract subclass

Interfaces



- What is the difference between an abstract class and an interface?
  - Java abstract class
    - Can have instance metho viour
    - May contain non-final va
  - Java interfaces
    - Methods are implicitly abstract and cannot have implementations
    - Variables declared are by default final

Interfaces

- Some explanations from the internet
  - An interface is a contract: the guy writing the interface says, "hey, I accept things looking that way", and the guy using th write looks that way".
  - An interface is an empty shell, there are only of the methods, which implies that the methods do not have a body. The int anything. It's just a pattern.
  - Abstract classes look a lot like interfaces, but they have something more: you can define a behavior for them. It's more about a guy saying, "these classes should look like that, and they have that in common, so fill in the blanks!".

Reference: http://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class

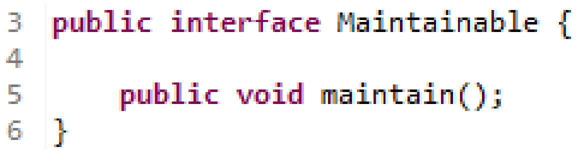
Interfaces

- Interfaces are less restrictive when it comes to inheritance
  - While classes can only ever extend one other class (single inheritance), with interfaces we can choose to implement a
  - Implementing an interface means writing imp ode for each of the methods in the interface

Interfaces

- Some rules:
  - Use the keyword "interface" instead of "class" to declare an interface
  - Implement an interface wit
  - Because interfaces have n hod actions, using an action name (ending in "able") is often appropriate
  - A class that implements an interface must pro tations for all the methods in the interface
  - Similar to classes, you can build up inheritance hierarchies of interfaces by using the "extends" keyword

Interfaces



```
public interface Maintainable {

    public void maintain();
}
```

Interfaces

Useful Website

https://www.tutorialspoint.com/java/

And finally …

Acknowledgement

- Slides based on material from
  - Bill Leahy's lecture slides
    - http://www.cc.gatech.ed oly.ppt
  - Maria Litvin's & Gary Litvin
    - http://skylit.com/javamethods/ppt/Ch10.ppt
  - Marty Stepp's lecture slides
    - http://www.cs.washington.edu/331/
- and others ...