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

Workshop 01 (with answers)

OO and Java Refresher (1/2)

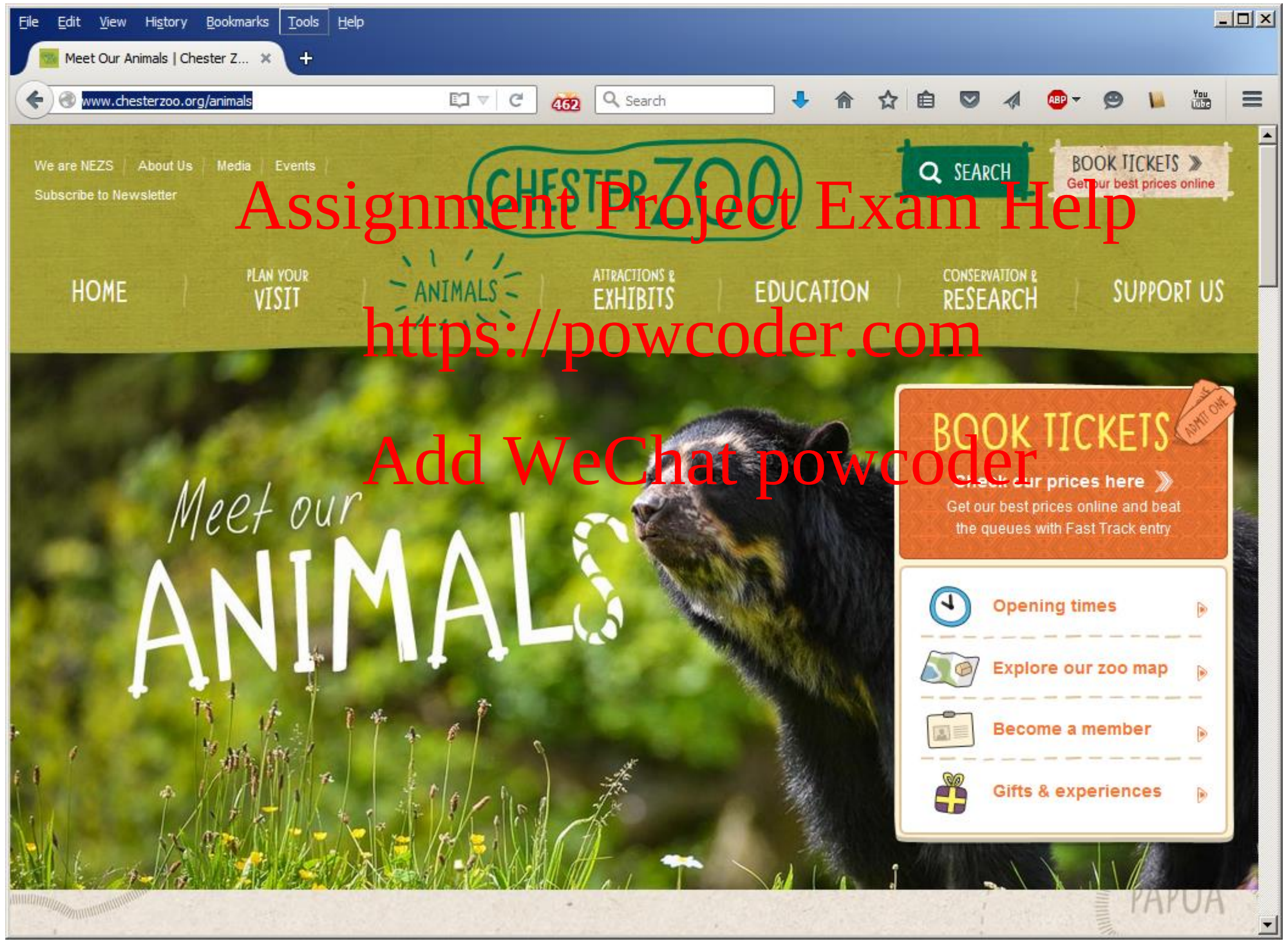
Peer-Olaf Siebers

University of Nottingham
UK | CHINA | MALAYSIA

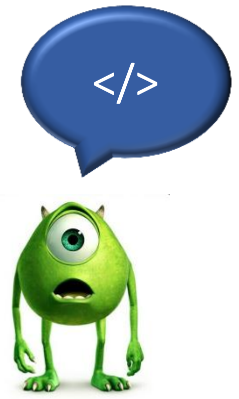
Topics

- Lecture 1
  - What is Software Maintenance?
  - Information about module organisation
  - Some examples of software maintenance challenges
- Lab 1
  - Eclipse and IntelliJ
  - Practicing Java basics
  - Working with existing code
- Workshop 1
  - OO and Java Programming Refresher

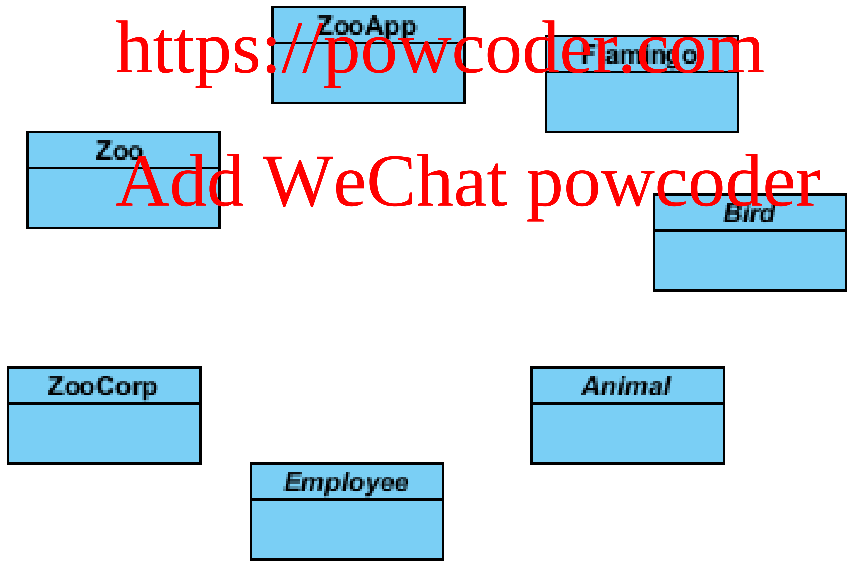
Case Study: Zoo Management

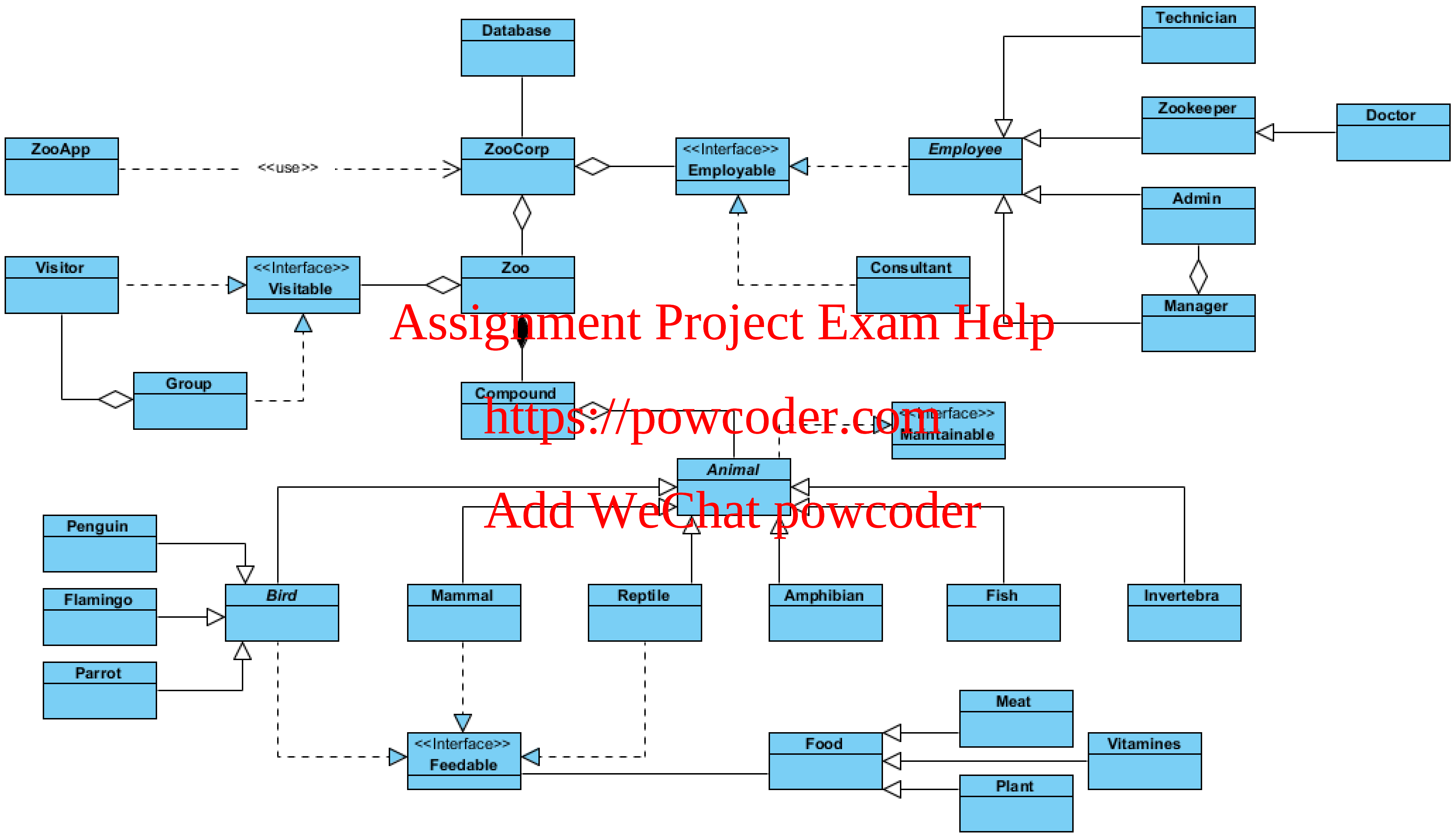


Case Study: Zoo Management



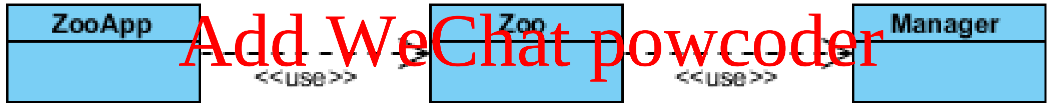
- Come up with a draft class diagram
  - Note that this is only a small choice of relevant classes!




Database
ZooApp
<<use>>
ZooCorp
<<Interface>> Employable
Employee
Technician
Zookeeper
Doctor
Admin
Manager
Consultant
Visitor
<<Interface>> Visitable
Zoo
Group
Compound
<<Interface>> Maintainable
Animal
Penguin
Flamingo
Parrot
Bird
Mammal
Reptile
Amphibian
Fish
Invertebra
<<Interface>> Feedable
Food
Meat
Vitamines
Plant

Case Study: Zoo Management

As we focus on Java basics today we want to keep it simple ...



Basic OO Concepts

</>

- Object-oriented programming is founded on these ideas:
  - **Abstraction**: Simple things like objects represent more complex underlying code and data
    - A class is a blueprint for a category of objects
    - An object is an entity that combines data with behavior that acts on that data
  - **Encapsulation** (information hiding): The ability to protect some components of the object from external access
    - e.g. keeping fields within a class private, then providing access to them via public methods
  - **Inheritance**: The ability for a class ("subclass") to extend or override functionality of another class ("superclass")

Basic OO Concepts

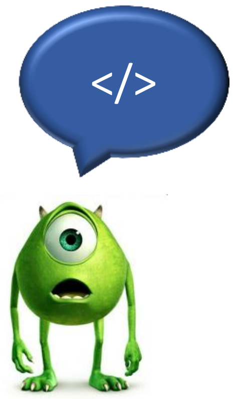
- Object-oriented programming is founded on these ideas:
  - **Polymorphism**: The provision of a single interface to entities of different types
    - Compile time (static) polymorphism through...
      - Method overloading: Create multiple methods with same name but different signatures
    - Run time polymorphism through...
      - Method overriding: Create method in derived class with same name and signature than in base class
      - Sub classing: reference of base class is able to reference, instantiate and destroy objects of derived class
  - **Interface**: A specification of method signatures (without implementations) as a mechanism for enabling polymorphism in a declarative way.

What's coming up ...

</>

- Public vs. Private
- Accessors and Modifiers
- Encapsulation
- The "this" keyword
- Constructors
- Passing parameters
- Static fields and methods

Public vs. Private



- What are the general rules for constructors, methods, helper methods, fields, and static constants?

  - Constructors and methods
    - Usually declared public (they constitute the interface of a class)

  - Helper methods that are needed only inside the class
    - Usually declared private

  - Fields
    - Usually declared private (to support encapsulation)
  - Static constants
    - Usually declared public

Accessors and Modifiers

- Accessors (also called Getters):
  - Methods that return values of private fields
  - Name often starts with get

- Modifiers (also called Mutators or Setters):
  - Methods that set values of private fields
  - Name often starts with set

Encapsulation

- Hiding the implementation details of a class (making all fields and helper methods private) is called encapsulation

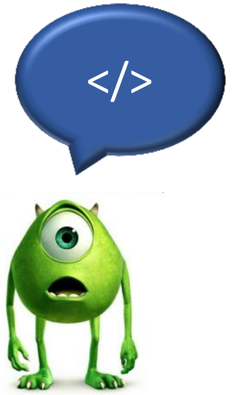
- Encapsulation helps in program maintenance: a change in one class does not affect other classes

- A client of a class interacts with the class only through well-documented public constructors and methods; this facilitates team development

The Keyword "this"

- "this" refers to the implicit parameter inside your class
  - A variable that stores the object on which a method is called
  - Refer to a field
    - this.field
  - Call a method
    - this.method(parameters);
  - One constructor can call another
    - this(parameters);

Constructors



- What are constructors used for?

Constructors

</>

- A constructor is a procedure for creating objects of the class
  - A constructor often initialises an object's fields
  - Constructors do not have a return type
  - All constructors in a class have the same name (the name of the class)
  - Constructors may take parameters
  - If a class has more than one constructor, they must have different numbers and/or types of parameters (constructor overloading)

- Important!
  - Java provides a default constructor for a specific class
  - If you define a constructor for a class, Java does not provide the default constructor anymore

Constructors

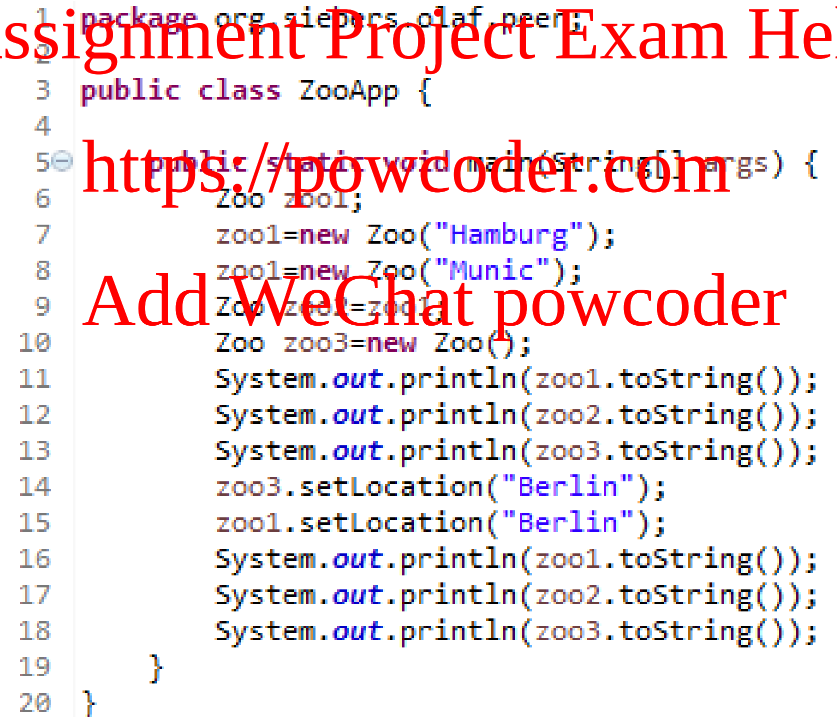
- Constructors of a class can call each other using the keyword "this" (referred to as constructor chaining) - a good way to avoid duplicating code

Invoking Constructors

- Constructors are invoked using the operator new.
  - Declare a reference variable of the required type and then invoke the constructor method after the "new" keyword.
- Parameters passed to "new" must match the number, types, and order of parameters expected by one of the constructors.

Invoking Constructors

- What does the output look like?



```
package org.siebers.olaf.peer;

public class ZooApp {

    public static void main(String[] args) {
            Zoo zoo1;
            zoo1=new Zoo("Hamburg");
            zoo1=new Zoo("Munic");
            Zoo zoo2=zoo1;
            Zoo zoo3=new Zoo();
            System.out.println(zoo1.toString());
            System.out.println(zoo2.toString());
            System.out.println(zoo3.toString());
            zoo3.setLocation("Berlin");
            zoo1.setLocation("Berlin");
            System.out.println(zoo1.toString());
            System.out.println(zoo2.toString());
            System.out.println(zoo3.toString());
    }
}
```

Passing Parameters

- In Java, parameters sent to methods are passed by value
  - Just to clarify some terminology
    - The "type" of data that a method can receive is referred to as a "parameter"
    - What is passed "to" a method is referred to as an "argument"
- Meaning of "pass-by-value"
  - In this case actual parameter is evaluated and its value is copied into memory (stack) used by the parameters of the method.
- Common misconception: "In Java primitives are passed by value and objects are passed by reference"
  - Objects are not passed by reference but object references (pointers) are passed by value
  - You can test this by using the "Litmus" test (writing a simple swap() function)

http://www.javadude.com/articles/passbyvalue.htm

Passing Parameters

- Inside a method, "this" refers to the object for which the method was called. "this" can be passed to other constructors and methods as a parameter.

Return Statement

- A void method can use a return statement to quit the method early
- There is no need for a return at the end

Overloaded Methods

- Methods of the same class that have the same name but different numbers or types of parameters are called overloaded methods
- The compiler treats overloaded methods as completely different methods
- The compiler knows which one to call based on the number and the types of the parameters passed to the method
- The return type alone is not sufficient for making a distinction between overloaded methods

Static Fields

- A static field (class field or class variable) is shared by all objects of the class
- A non-static field (instance field or instance variable) belongs to an individual object

- Static fields are stored with the class code, separately from instance variables that describe an individual object

Static Fields

</>

- Public static fields, usually global constants, are referred to in other classes using dot notation
  - ClassName.constName
- Usually static fields are NOT initialized in constructors (they are initialized either in declarations or in public static methods).
- If a class has only static fields, there is no point in creating objects of that class (all of them would be identical).
  - Math and System are examples of the above (they have no public constructors and cannot be instantiated)

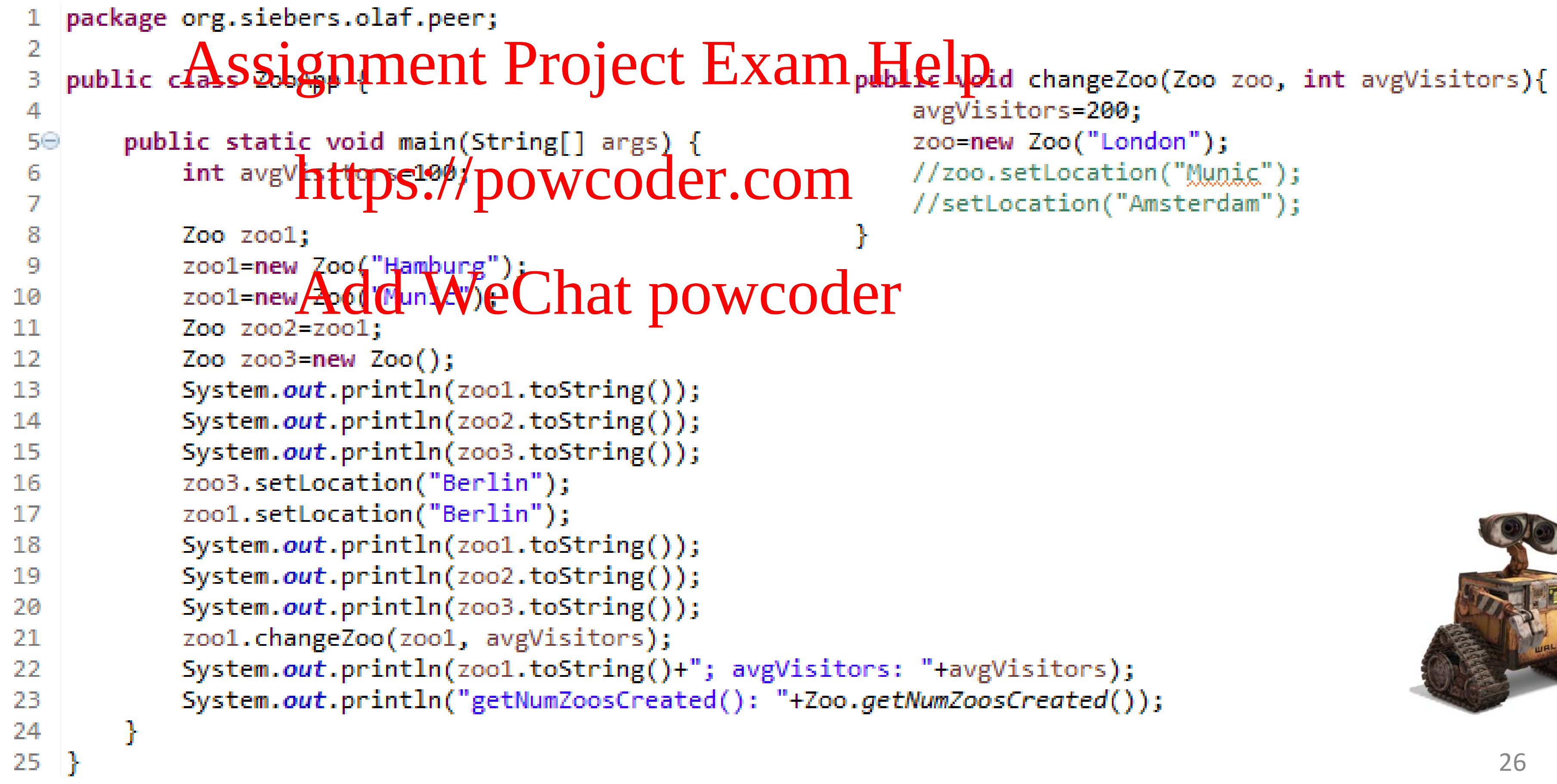
Static Methods

- Static methods can access and manipulate class's static fields. They belong to the class - not an instance of it.
- Static methods cannot access instance fields or call instance methods of the class; instance methods can access all fields and call all methods of their class - both static and non-static
- Static methods will usually take input from the parameters, perform actions on it, then return some result.

- Static methods are called using dot notation
  - ClassName.statMethod(...)

Static Fields and Methods

- What does the output look like?

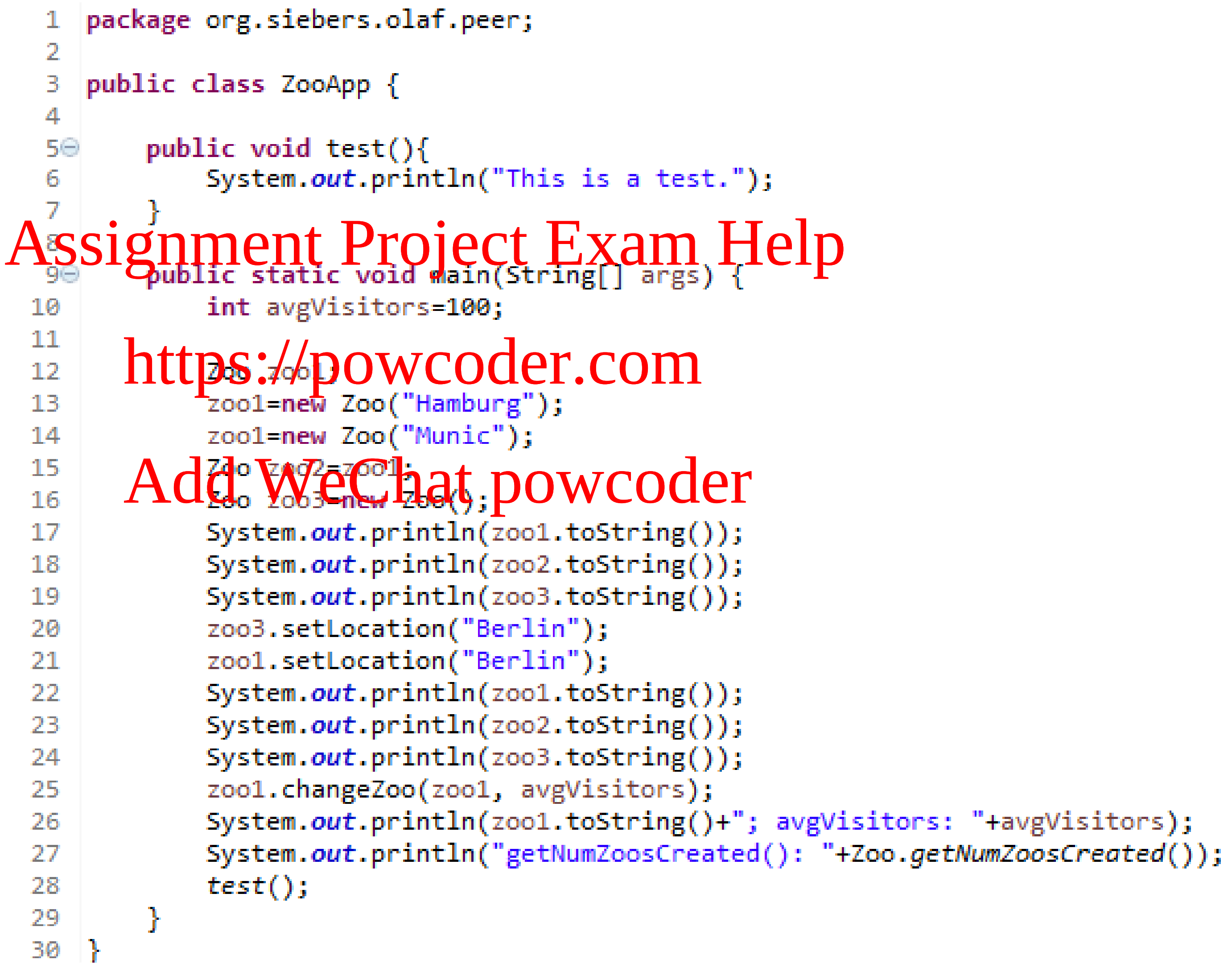


```
package org.siebers.olaf.peer;

public class ZooApp {

    public static void main(String[] args) {
        int avgVisitors=100;

        Zoo zoo1;
        zoo1=new Zoo("Hamburg");
        zoo1=new Zoo("Munic");
        Zoo zoo2=zoo1;
        Zoo zoo3=new Zoo();
        System.out.println(zoo1.toString());
        System.out.println(zoo2.toString());
        System.out.println(zoo3.toString());
        zoo3.setLocation("Berlin");
        zoo1.setLocation("Berlin");
        System.out.println(zoo1.toString());
        System.out.println(zoo2.toString());
        System.out.println(zoo3.toString());
        zoo1.changeZoo(zoo1, avgVisitors);
        System.out.println(zoo1.toString()+"; avgVisitors: "+avgVisitors);
        System.out.println("getNumZoosCreated(): "+Zoo.getNumZoosCreated());
    }
}
```

```
public void changeZoo(Zoo zoo, int avgVisitors){
    avgVisitors=200;
    zoo=new Zoo("London");
    //zoo.setLocation("Munic");
    //setLocation("Amsterdam");
}
```

Static Fields and Methods

- Does this compile?



```java
package org.siebers.olaf.peer;

public class ZooApp {

    public void test(){
        System.out.println("This is a test.");
    }

    public static void main(String[] args) {
        int avgVisitors=100;

        Zoo zoo1;
        zoo1=new Zoo("Hamburg");
        zoo1=new Zoo("Munic");
        Zoo zoo2=zoo1;
        Zoo zoo3=new Zoo();
        System.out.println(zoo1.toString());
        System.out.println(zoo2.toString());
        System.out.println(zoo3.toString());
        zoo3.setLocation("Berlin");
        zoo1.setLocation("Berlin");
        System.out.println(zoo1.toString());
        System.out.println(zoo2.toString());
        System.out.println(zoo3.toString());
        zoo1.changeZoo(zoo1, avgVisitors);
        System.out.println(zoo1.toString()+"; avgVisitors: "+avgVisitors);
        System.out.println("getNumZoosCreated(): "+Zoo.getNumZoosCreated());
        test();
    }
}
```

And finally ...

References

- Sommerville (1992) 'Software Engineering' 4e, Pearson.

</>

Acknowledgement

- Slides based on material from
  - Bill Leahy's lecture slides
    - http://www.cc.gatech.edu/~bleahy/xjava/cs1311xjava05_poly.ppt
  - Maria Litvin's & Gary Litvin's book slides
    - http://skylit.com/javamethods/ppt/Ch10.ppt
  - Marty Stepp's lecture slides
    - http://www.cs.washington.edu/331/
  - And others ...

But I also contributed some stuff myself :-)