LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Fakultät für Betriebswirtschaft
Munich School of Management

Basics in Programming for MMT

Session 8 – Interfaces

MMT

www.mmt.bwl.lmu.de

Scope of the Session

1. Repetition

- Inheritance

2. Theory

- Interfaces
- Design Patterns

3. Assignment

- Assignment
- What to implement?

Repetition

Repetition

Inheritance (1/2)

- Inheritance enables us to define functionalities and data structures once that are shared by different classes.
- The child class (`B`) inherits all the **fields** and **methods** of its parent class (`A`).

```
class A {
  float x;
  ...
  void doSomething () {...}
}

class B extends A {
  // float x;
  ...
  // void doSomething () {...}
}

A a;
B b;

void setup () {
  a = new A();
  b = new B();
  a.doSomething();
  b.doSomething();
}
```

Repetition

Inheritance (2/2)

Keyword `extends` + parent name

```
class B extends A {
```

Fields

```
  float x;
  float y;
  // fields of A are inherited
```

Constructor: call parent constructor

```
  public b (int x, int y) {
    super(...); // call A constructor
    this.x = x;
    this.y = y;
  }
```

Methods

```
  void doSomething () {...}
  // methods of A are inherited
```

End of class

```
}
```

Theory

Theory

Interfaces (1/2)

- Even if some classes do not share the same parent class, they could include the same methods.
- These methods could further differ in their implementations.

```
class A {
  ...
  void doSomething () {}
}

class Z {
  ...
  void doSomething () {}
}
```

Theory

Interfaces (2/2)

- Interfaces define methods that have to be implemented by the classes implementing this interface.
- This is not optional!
- The interface specifies the **return type**, the **name** and the **arguments**, but not the commands that are performed.

```
interface Doable {
  void doSomething ();
  void doSomethingElse (int x);
}

class A implements Doable {
  ...
  void doSomething () {...}
  void doSomethingElse (int x) {...}
}

class Z {
  ...
  void doSomething () {...}
  void doSomethingElse (int x) {...}
}
```

Theory

Design Patterns

- This is used to “force” programmers to implement specific functions if they used third party code/libraries.
- Often these make use of **design patterns**.
- Design patterns are program structures that are useful to solve certain recurring problems.

```
interface Doable {
  void doSomething ();
  void doSomethingElse (int x);
}

class A implements Doable {
  ...
  void doSomething () {...}
  void doSomethingElse (int x) {...}
}

class Z {
  ...
  void doSomething () {...}
  void doSomethingElse (int x) {...}
}
```

Assignment

Assignment

- You have to implement additional functionalities (min. 4) for our PONG game.
- These changes have to go beyond “style” changes: color, size, etc.
- Do not copy the code of others!

Assignment

What to implement?

- **2D movement of Bats:** make the bats move in all directions
- **Multi Ball:** implement another ball, possibly based on certain events (e.g. score)
- **Moving obstacles:** implement moving obstacles that sometimes move over the screen and can cause balls to bounce
- **Score:** count the score for both players and display it
- **Color Changes:** make the colors change based on certain events (e.g. score)
- **Fast Ball:** if the ball does not leave the boarders, increase its speed constantly Reset the speed if one player scored
- **Shrinking Bats:** make the bats shrink over time to increase the difficulty of the game