Poznan University of Technology

Object Oriented Programming

Jakub Piotr Hamerliński, M.Eng.

Object Oriented Programming

Agenda

- Interfaces
- Immutability
- Task

Interfaces

Object Oriented Programming

Interfaces

Interface is a contract that object must obey. It consists of methods declarations.

```java
//java
interface Money {
  Money multiply(float factor);
  String balance();
}
```

```cpp
//cpp
class Money {
  public:
    virtual std::string balance() = 0;
};
```

Object Oriented Programming

Interfaces

```java
//java
interface Money {
  Money multiply(float factor);
  String balance();
}
```

```java
//java
public class Cash implements Money {
  private final float dollars;
  @Override
  public Cash multiply(float factor) { return new Cash(this.dollars * factor); }
  @Override
  public String balance() { return "$" + dollars; }
  public Cash(float dollars) { this.dollars = dollars; }
}
```

```java
//java
class Employee {
  private Money salary;
}
```

Object Oriented Programming

Interfaces

```cpp
//cpp
class Money {
public:
 virtual std::string balance() = 0;
};
```

```cpp
//cpp
class Cash: public Money {
private:
    float dollars;
public:
    Cash multiply(float factor){
      return Cash(dollars * factor);
    }
    std::string balance(){
      //some code to achieve float with proper precision(...)
      return "$" + s;
    }
    Cash(float dollars){
      this->dollars = dollars;
    }
};
```

Object Oriented Programming

Interfaces

The rule here is simple: every public method in a good object should implement his counterpart from an interface. If your object has public methods that are not inherited from any interface, he is badly designed. There are two practical reasons for this. First, an object working without a contract is impossible to mock in a unit test. Second, a contract-less object is impossible to extend via decoration.

Class exists only because someone needs its service. The service must be documented - it's a contract, an interface.

Immutability

immutability (noun, BrE /ɪˌmjuːtəˈbɪlətɪ/):
the fact of never changing or being changed

Oxford Dictionary

Object Oriented Programming

Immutability

A good object should never change his encapsulated state.
Bad example:

```
1  //java
2  class Cash {
3    private float dollars;
4    public void multiply(float factor) { this.dollars *= factor; }
5  }
6
7  Cash five = new Cash(5.00f);
8  five.multiply(10.00f);
9  System.out.println(five.balance()); // oops! "$50.00" will be printed!
```

Object Oriented Programming

Immutability

A good object should never change his encapsulated state.
Good example:

```
//java
class Cash {
  private final float dollars;
  public Cash multipliedCash(float factor) { return new Cash(this.dollars * factor); }
}

Cash five = new Cash(5.00f);
Cash fifty = five.multipliedCash(10.00f);
System.out.println(fifty.balance()); // "$50.00" will be printed :)
```

Object Oriented Programming

Immutability

A good object should never change his encapsulated state.
Bad example:

```cpp
//cpp
class Cash {
  private:
    float dollars;
  public:
    void multiply(float factor) { dollars *= factor; }
}

Cash five(5);
five.multiply(10);
std::cout<<five.balance(); // oops! "$50.00" will be printed!
```

Object Oriented Programming
Immutability

A good object should never change his encapsulated state.
Good example:

```cpp
//cpp
class Cash {
  private:
    float dollars;
  public:
    Cash multipliedCash(float factor) { return Cash(this.dollars * factor); }
}

Cash five(5.00f);
Cash fifty = five.multipliedCash(10.00f);
std::cout<<fifty.balance(); // "$50.00" will be printed :)
```

Object Oriented Programming

Immutability

A good object should never change his encapsulated state.
Good example:

```java
//java
final class HTTPStatus implements Status {
  private URL page;
  public HTTPStatus(URL url) {
    this.page = url;
  }
  @Override
  public int read() {
    return HttpURLConnection.class.cast(
      this.page.openConnection()
    ).getResponseCode();
  }
}
```

Object Oriented Programming

Immutability

- Immutable objects are simpler to construct, test, and use.
- Truly immutable objects are always thread-safe.
- They help avoid temporal coupling.
- Their usage is side-effect free.
- They are much easier to cache.
- They prevent NULL references.

Object Oriented Programming

Task

Write an **interface** or **abstract class** and a **class** in Java or C++, which will be immutable and will implement that **interface** or **abstract class**.

Questions?

Fin