Immutability & Copy-in-copy-out

CSE 373 Help Section

Interface v.s Implementation

- Interface: what people see when use it

  e.g. pop() means to return the last element in a stack

- Implementation: how we store and process internally

  e.g. use array or linked list to store elements in a stack

Preserving Abstraction

```
public class Queue {
    public Person[] people;
    ...
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...

    public void insert(Person p) {
        … // put p into people
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...

    public void insert(Person p) {
        Person internalP = new Person(
            p.getName(), p.getAge(), p.getContact());
        … // put internalP into people
    }
}
```

Copy in!

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    …

    public void insert(Person p) {
        ContactInfo c = p.getContact();
        ContactInfo internalContact = new ContactInfo(
            c.getaddress(), c.getPhoneNumber());
        Person internalP = new Person(
            p.getName(), p.getAge(), internalContact);
        … // put internalP into people
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Deep copy in!

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    …

    public void insert(Person p) {
        ContactInfo c = p.getContact();
        ContactInfo internalContact = new ContactInfo(
            c.getaddress(), c.getPhoneNumber());
        Person internalP = new Person(
            p.getName(), p.getAge(), internalContact);
        … // put internalP into people
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Good copy in.

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    …

    public Person peek() {
        … // find and return the first person
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    …

    public Person peek() {
        … // find the first person p
        return new Person(
            p.getName(), p.getAge(), p.getContact());
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Copy out!

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    …

    public Person peek() {
        … // find the first person p
        ContactInfo c = p.getContact();
        ContactInfo externalContact = new ContactInfo(
            c.getaddress(), c.getPhoneNumber());
        return new Person(
            p.getName(), p.getAge(), externalContact);
    }
}
```

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Deep copy out!

Preserving Abstraction

```
public class ContactInfo {
    private String address;
    private String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

```
public class Queue {
    private Person[] people;
    ...

    public Person peek() {
        … // find the first person p
        ContactInfo c = p.getContact();
        ContactInfo externalContact = new ContactInfo(
            c.getaddress(), c.getPhoneNumber());
        return new Person(
            p.getName(), p.getAge(), externalContact);
    }
}
```

Good copy out.

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...

    public void insert(Person p) {
        … // put p into people
    }

    public Person peek() {
        … // find and return the first person
    }
}
```

```
public class ContactInfo {
    private final String address;
    private final String phoneNumber;
}
public class Person {
    private final String name;
    private final int age;
    private final ContactInfo contact;
}
```

Good protection.

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...

    public void insert(Person p) {
        … // put p into people
    }

    public Person peek() {
        … // find and return the first person
    }
}
```

```
public class ContactInfo {
    private final String address;
    private final String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

Preserving Abstraction

```
public class Queue {
    private Person[] people;
    ...

    public void insert(Person p) {
        Person internalP = new Person(
            p.getName(), p.getAge(), p.getContact());
        … // put internalP into people
    }

    public Person peek() {
        … // find the first person p
        return new Person(
            p.getName(), p.getAge(), p.getContact());
    }
}
```

```
public class ContactInfo {
    private final String address;
    private final String phoneNumber;
}
public class Person {
    private String name;
    private int age;
    private ContactInfo contact;
}
```

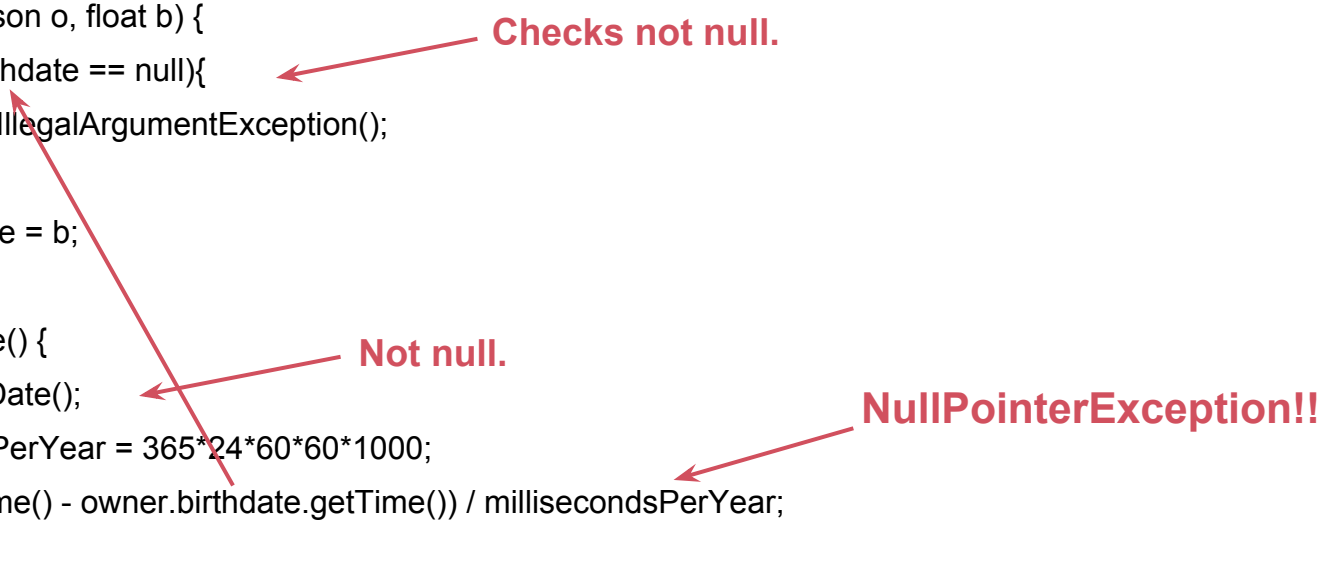
Copy in!

Good enough.

Copy out!

14wi Final

```
class BankAccount {
        private Person owner;
        private float balance;
        public BankAccount(Person o, float b) {
                if(o == null || o.birthdate == null){
                        throw new IllegalArgumentException();
                }
                owner = o; balance = b;
        }
        public long getOwnerAge() {
                Date now = new Date();
                long millisecondsPerYear = 365*24*60*60*1000;
                return (now.getTime() - owner.birthdate.getTime()) / millisecondsPerYear;
        }
}
```



14wi Final

```
class BankAccount {
        private Person owner;
        private float balance;
        public BankAccount(Person o, float b) {
                if(o == null || o.birthdate == null){
                        throw new IllegalArgumentException();
                }
                owner = o; balance = b;
        }
        public long getOwnerAge() {
                Date now = new Date();
                long millisecondsPerYear = 365*24*60*60*1000;
                return (now.getTime() - owner.birthdate.getTime()) / millisecondsPerYear;
        }
}
```

```
Person p = new Person();
p.name = "Bob";
p.birthdate = new Date(1988, 10, 17);
BankAccount acct = new BankAccount(p, 10.0);
p.birthdate = null;
acct.getOwnerAge();
```

Fixation:
The constructor of BankAccount should do a deep copy of the Person object passed in.