Career: Ingeniería en Telecomunicaciones
Date: 07-07-2021
Subject: OBJECT ORIENTED PROGRAMMING
NRC: 3730
Teacher: ING. JORGE LASCANO, PHD, MS
Names:

1. Sunta Barzallo, Josselyn Gabriela
2. Taco Loachamin, Lizeth Carolina
3. Titoaña Tigse, Leslie Brigitte
4. Tupiza Portilla, Solange Anahi
5. Villavicencio Antamba, Alina Patricia
6. Yanez Loaiza, Erick Dario
7. Yugsi Tapia, Maria Pamela

Abstraction

Background

- From a process perspective, abstraction is the act of bringing certain details to the forefront while suppressing all others.
- An abstraction is anything that exposures certain details that others can use and rely on.
- Some artifacts define their “public” details explicitly; others do not.
- Software abstraction requires developers to sift through large and diverse collections of details, and then determine the most salient and distinguishing concepts.

EXAMPLE - ABSTRACTION OF “PERSONS”

- List all the properties of persons who might be interesting to football (soccer) tournament tracking system.

Fast Tall Age Coordination Physical aptitude Weight Gender Equipment

TWO COMMON PROBLEMS WITH ABSTRACTION

- **Leaky Abstraction:**
    - External characteristics are not defined as such.
    - This problem can be solved with good encapsulation.

- **Over Abstraction:**
    - Insufficient control given to users of the abstraction.
    - Inadequate access to the information embodied in the abstraction.

ABSTRACTION

Abstraction is the act of summarizing or generalizing something to focus on the ideas most relevant to a conversation or certain kind of communication.

In object orientation, abstraction (the verb) is the creation of interfaces of a components, i.e., a class that exposes certain details necessary for working with that component.

An abstraction (the noun) is a description that leaves out unnecessary details.

“interfaces”, as found in C# and Java, are abstractions, but so are
abstract classe
Adherence Criteria:

- Meaningful labels and identifiers
- Context-aware labels and identifiers
- Explicit declaration of the full abstraction
- Abstraction sufficiency (completeness)
- Abstraction conciseness (non-redundancy)

MEANINGFUL LABELS AND IDENTIFIERS

- A good name must be precise, it must not be too wide or too narrow, the way of speaking must be appropriate.
- Any kind of variable name, method name, etc.) are critical to good abstraction because they impact understanding.
- Class and variable names should be nouns or noun phrases and Methods names should be verbs.

CONTEXT - AWARE LABELS AND IDENTIFIES

- Labels and identifiers should be redundant within the context they are declared and most often references.
- Examples from “Contacts”
- See person-only (java)/good
- See person-only (java)/poor abstraction – names not context-aware

EXPLICIT DECLARATION OF AN ABSTRACTION

• In object-oriented (OO) languages, the public interface for a class consists
of all the public elements (e.g., data members, methods, inner classes,

etc.)

• Some aspects of how developers use objects of a particular class cannot be captured in a class definition, e.g.,

- Constraints of the order in which methods must be called
- Constraints of specializations

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function.

A pure virtual function is one which must be overridden by any concrete (i.e., non-abstract) derived class. This is indicated in the declaration with the syntax " = 0" in the member function's declaration.

The following is an example of an abstract class:

```
class AB {
        public:
        virtual void f() = 0;
}
```

• accidently exposed, e.g., when a method returns a modifiable object returns that is

ABSTRACTION SUFFICIENCY

- Objects (instances of classes) are intended to be used.
- A class's abstraction represents what and how things can be done with an object.
- This adherence criterion requires the developer to ensure that users of a class can do what they need to do with the objects of that class.

ABSTRACTION CONCISENESS

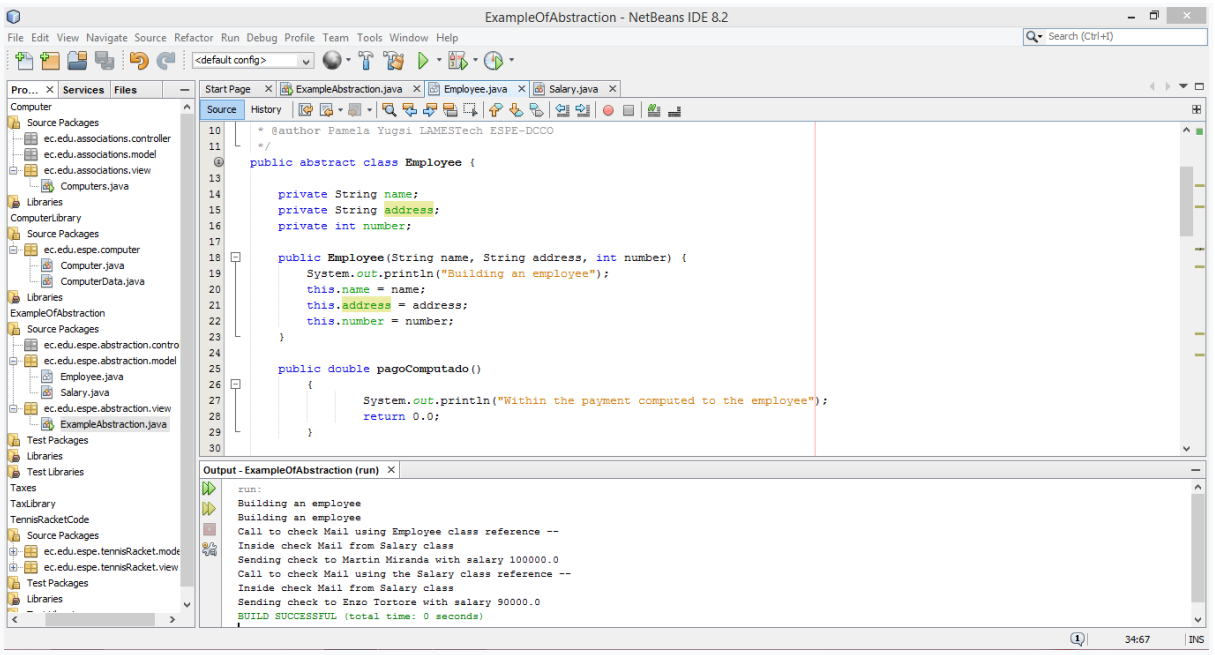
- To reduce complexity and increase maintainability, abstractions should not contain redundant mechanisms for the same things unless they are methods.
- The methods are either required for compatibility with legacy code or are also required form an adaptor.

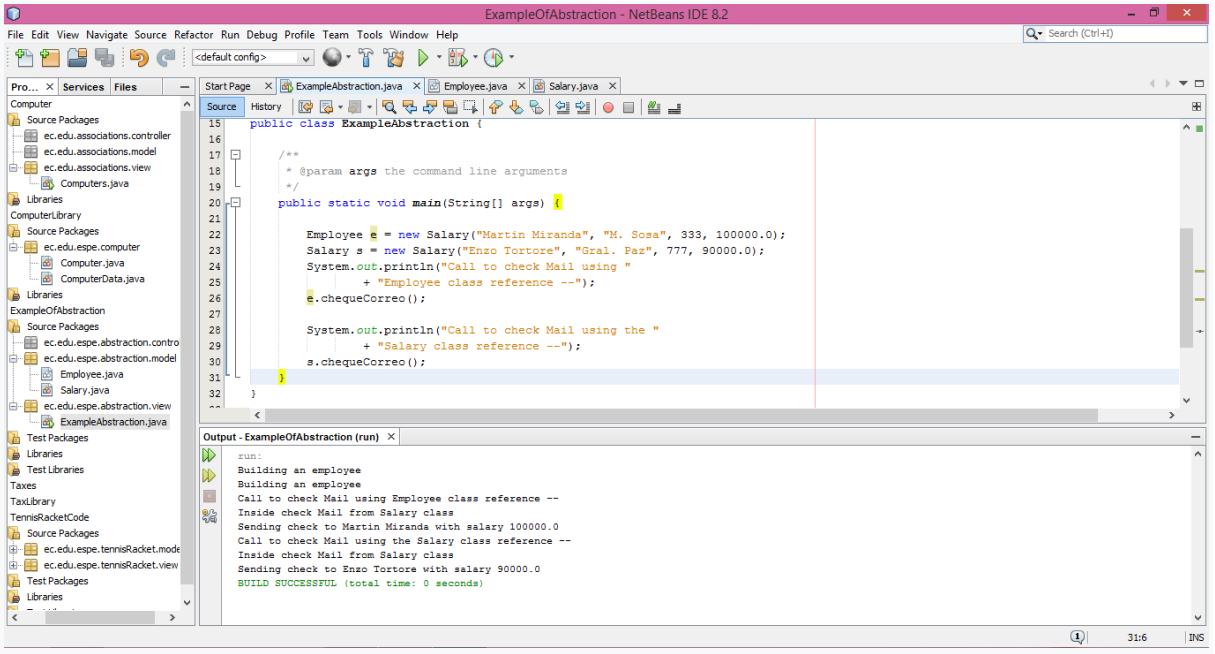
Example of Abstraction Conciseness

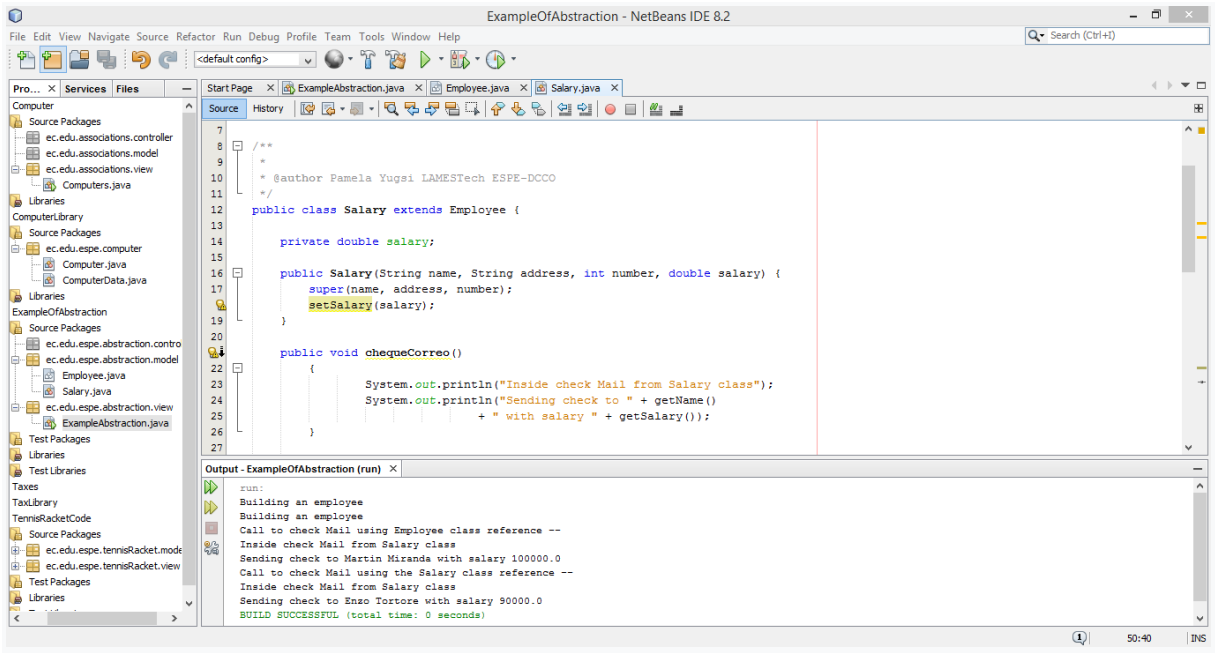
```
from abc import ABCMeta, abstractmethod

class AbstractFoo:
     __metaclass__ = ABCMeta

     @abstractmethod
     def bar(self):
     pass

     @classmethod
```

```
        def __subclasshook__(cls, C):
        return NotImplemented


class Foo(object):
    def bar(self):
    print "hola"


AbstractFoo.register(Foo)
```

EXAMPLE







QUESTIONS

1. What is abstraction?
a. It is based on another object or class, using the same implementation or behavior.
b. The act of bringing certain details to the forefront while suppressing all others.
c. Minimize the domino effect when software changes occur in expected (and even some unexpected) ways.

2. A ……. includes all that is exposed to the users of the class, regardless of whether something is exposed.
a. Developer.
b. Language.
c. Class's Abstraction.

3. A class's abstraction represents what and how things can be done with an .....
a. Object.

b. Class.
c. Method.

4. Abstraction: Adherence criteria
a. redundancy.
b. Context-aware labels and identifiers.
c. labels and identifiers without text.

5. What is the difference between an abstract class and a normal class?
a. An abstract class must have at least one abstract method
b. An abstract class must have at least three conventional attributes
c. An abstract class and a normal class differ in their methods and attributes