Relationships between objects I

**Christian Rodríguez Bustos**

Object Oriented Programming



*Relationships between objects*

**

Agenda

Static Keyword

Relationships between

objects

Inheritance Inheritance and Java I



*Relationships between objects*

static Keyword

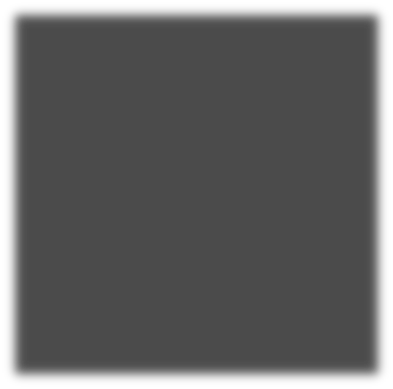
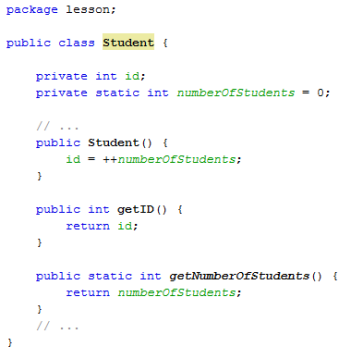
Static attributes

Static methods

*Relationships between objects*

**

Static attributes

Static attributes are 

**common to all**

**instanced** objects

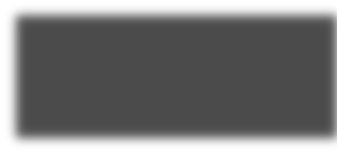
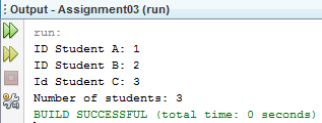
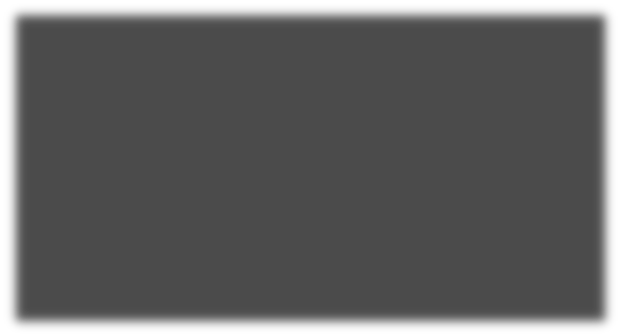
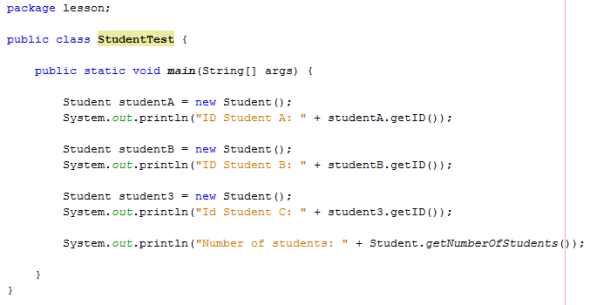
Static attributes are

**class attributes**

*Relationships between objects*

**

Static attributes example



*Relationships between objects*

**Student A Id = 1

Static attributes example **Static property** Number of students = **1**

*****Relationships between objects*

Static attributes example



Student A Id = 1

Student B

Id = 2

**Static property** Number of students = **2**

*****Relationships between objects*

Static attributes example



Student A Id = 1

Student B

Id = 2

**Static property** Number of students = **3**

Student C Id = 3

*Relationships between objects*

Accessing static attributes

*Relationships between objects*

Accessing static methods or attributes

We **do not need to instantiate**

**any object** to access static

attributes or methods





*Relationships between objects*

Examples

Static methods typically take all they data from parameters and compute something from those parameters.



*Relationships between objects*

TicTacToe Examples











*Relationships between objects*

Accessing static methods or attributes

Be careful!!!

Static methods cannot access Non-Static attributes







*Relationships between objects*

Relationships between objects

Association and links

Aggregation and composition

UML notation



Associations are structural relationship

Associations are 

structural

relationship that

exists between

**classes**

• A **Professor** *teaches* one

or many **Groups**

• A **Course** *is offered* as

one or many **Groups**

• Zero or many **Students**

*attends* one or many

**Groups ***Relationships between objects*

Links

Are relations between two specifics **objects** (*instances*)

**Association** :

*attends at*

A student Any group

**Link**: *attends at*

Bruce Wayne Math 3B

A ***specific*** student A ***specific*** group *Relationships between objects*

**

Higher order associations

One or more **Student** *attends*

one or more **Groups**

One or more **Students** *receive* one or more **Grades**

One **Group** *issues* one or more **Grades**

****Student 

Group Grade



*Relationships between objects*

Higher order associations

Higher order associations are represented by two classes associations





*Relationships between objects*

Aggregation and Composition

**Aggregation:** Is a specific type of association, is

represented typically by “*consists of*”, “*is composed of*” and “*has a*”

**Composition:** Is a strong form of aggregation, in which **the “parts” cannot exist without the “whole.”**

A Team ***is composed by*** one or more Students

A Department ***is composed of*** one or more Professors

A Club ***has*** Members

A Building ***is composed by*** one or more Rooms

A University ***is composed of*** Departments A Board ***is composed of*** Squares



*Relationships between objects*

Aggregation code example



Class definition 

A Team ***is composed by*** one or

more Students



Test code If Team is destroyed, the Students **still** exist

*Relationships between objects*

**

Composition code example

A Board ***is composed of*** Squares

Test code

Class definition

If Board is destroyed, the squares are destroyed too



*Relationships between objects*

UML notation



Aggregation is depicted as an **unfilled diamond**

Composition is

depicted as a **filled**

**diamond** and a solid

line.



*Relationships between objects*

Inheritance

Hierarchy of classes

UML Notation



Actual Situation

Typical Student has an 

id and a user

We @override **toString**

method to print the

Student data

*Relationships between objects*

New Requirements arrives

The Academic

Information System have

to manage the 

information of **graduate**

**students**:

– Undergraduate

program

– Current place of

employ

*Relationships between objects*

Your boss



**Solution 1**

Modify the Student Class *Relationships between objects*

**

Solution 1 - Modify the Student Class

We can **add new parameters, setters and getters and modify the toString method** to print depending of type of student



*Relationships between objects*

Solution 1 - Modify the Student Class

*Relationships between objects*

New Requirements arrives

The system must to

handle the graduate

program being taken by 

graduate students:

– Current graduate

program

Your boss

*Relationships between objects*

**

Solution 1 - Modify the Student Class again ???

You have to **add one more attribute, two methods and a extra validation** in the toString method



*Relationships between objects*

Solution 1 is a bad solution

Your student class is not 

well delimited, at this

moment **your objects**

**students can be**

**understood as**

**graduate and**

**undergraduate**.

How can we distinguish

between one or

another? *Relationships between objects*

**Solution 2**

Create GraduateStudent class *Relationships between objects*

**

Solution 2 – Create GraduateStudent class 

This makes sense, it could work!!!

*Relationships between objects*

Solution 2 – Create GraduateStudent class

Wait!!! , this code smells 

like a **cloned code!**

****

*Relationships between objects*

**

**Clones!!!**

*Relationships between objects*

**

Try to avoid cloning

Clones are hard to

maintain and reflect

poor designs.



*Relationships between objects*

**

**Solution 3**

Taking Advantage of Inheritance *Relationships between objects*

**

Solution 3 - Taking Advantage of Inheritance 

Gradate Student **inherit all accessible methods and attributes** from Student class

*Relationships between objects*

Inheritance terms

Graduate Student

is a **specialization** of Student

is a **subclass** of Student

Student

is a **generalization** of a Graduate Student

is the **superclass** of Student



*Relationships between objects*

Inheritance terms



**A** *Is a*

subclass superclass

A **Graduate Student** *is a* **Student**

A **Graduate Student** *is a specialization of a* **Student**

****

*Relationships between objects*

Class Hierarchies Generalization

Student

Person 

Professor

We manage knowledge in terms of inheritance hierarchies 

In a POO language we can abstract the real

Graduate Student

Specialization

world relation into **class**

**hierarchies**

****

*Relationships between objects*

Inheritance is one of the four principles of OOP Abstraction Encapsulation

Inheritance ?????????



*Relationships between objects*

Inheritance benefits

**Reduction of**

**code redundancy**

• Maintenance

• Avoid “Ripple Effects”

We can **reuse and extend** code that has already been tested

Subclasses are more **concise**

We can **derive** a new class from an existing class

Inheritance **is a natural way to manage**

**knowledge**

*Relationships between objects*

**

Class Hierarchies inevitably expand over time Person

Student

Graduate

Professor

Student Undergraduate *Relationships between objects*

**

Class Hierarchies inevitably expand over time Person

Student

Graduate

Professor Administrator

Student Undergraduate *Relationships between objects*

**

Class Hierarchies inevitably expand over time Person

Student

Graduate

Student

Masters PhD

Professor Administrator

Undergraduate



*Relationships between objects*

UML notation

• A Student is a Person • A Professor is a Person



*Relationships between objects*

Java Inheritance I

The object class



All classes are subclasses of the Object class

In Java **Class Object** is the

root of the class hierarchy.

Object

Every class has Object as a

superclass.

Person

String System

All objects, including arrays, inherit the methods of this class.

Student Professor



*Relationships between objects*

All classes are subclasses of the Object class



Is equivalent

to





*Relationships between objects*

**

All those methods are

inherited by all classes



*Relationships between objects*

Time to play

1. Using UML design a **class hierarchy** (at least 3 levels of inheritance) for a **pet store with** at least 6 different kind of pets

2. Create the **Java classes definitions** (encapsulated) for the pets available on the pet store, each pet must have at least 3 attributes (not inherited).



*Relationships between objects*

References

[Barker] J. Barker, *Beginning Java Objects: From Concepts To Code*, Second Edition, Apress, 2005.

[Oracle] *Understanding Instance and Class Members,* Available: http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html

[Oracle] Java API documentation, *Class Object,* Available: http://download.oracle.com/javase/6/docs/api/java/lang/Object.html



*Relationships between objects*