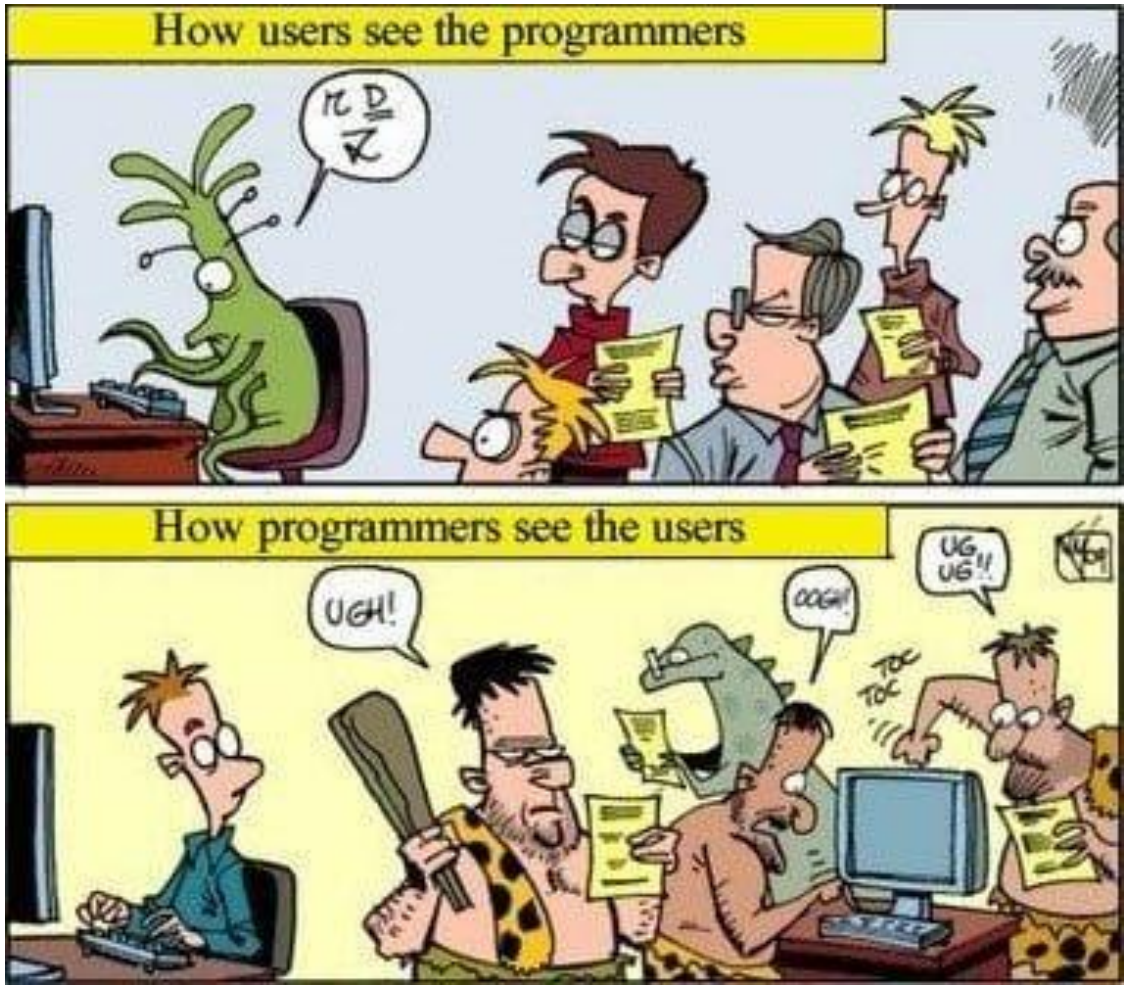

How users see the programmers
How programmers see the users
UGH!
OOGH!
UG UG!!
TOC TOC

Exam question samples

- Define and explain polymorphism. Give examples.
- What are differences between inheritance and composition?
- What are the ***this*** and ***super*** keywords used for?
- Explain how List can be sorted in java, take a concrete java example, give associated code and output.
- Explain why element deletion in LinkedList is faster than in ArrayList.
- What are Comparator and Comparable used for? Give examples.

PART 10

Abstraction

Abstract Class

- An abstract class is a class containing one or more abstract methods
- An abstract method is a method that will be fully defined in a derived class
- Syntax for defining an abstract method:

  public abstract returnType methodName(Parameters_List);

or:

  protected abstract returnType methodName(Parameters_List);

- An abstract method contains no code
- You could never create an instance of an abstract class

Abstract Class

```
public abstract class Character {
    protected String name;

    /** Character constructor, it is not possible ...5 lines */
    public Character(String name){...3 lines }

    /** Accessor of the name attribute ...4 lines */
    protected String getName(){...3 lines }

    /** method that allows the character to talk and ...5 lines */
    public void talk(String say){...3 lines }

    /** Each time a character is created, he/she has ...4 lines */
    public void introduceYourself(){...3 lines }

    public abstract String whoAreYou();
}
```

Abstract Class

```
/**
 *
 * @author mikae
 */
public class Lady extends Character{
    private String dressColor;
    private boolean isKidnapped;
    /** Lady constructor, it is not possible ...6 lines */
    public Lady(String name, String dressColor){...7 lines }

    /** Ladies introduce themselves in a very lady way ...3 lines */
    @Override
    public void introduceYourself() {...4 lines }

    /** Unfortunately in many westerns at some point, ladies are kidnapped by a  robber ...4 lines */
    public void hasBeenKidnapped(Robber robber){...5 lines }


}
```

Lady is not abstract and does not override abstract method whoAreYou() in Character
---
(Alt-Enter shows hints)

Abstract Class

```
public class Western {
    public static void main(String[] args) {

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        System.out.println("");

        Robber butch = new Robber("Butch Cassidy", "Wicked");
        System.out.println("");

        Lady sam = new Lady("Samantha", "Pink");
        System.out.println("");

        butch.kidnapLady(sam);


        Character newCharacter = new Character();

    }
}
```

Character is abstract; cannot be instantiated

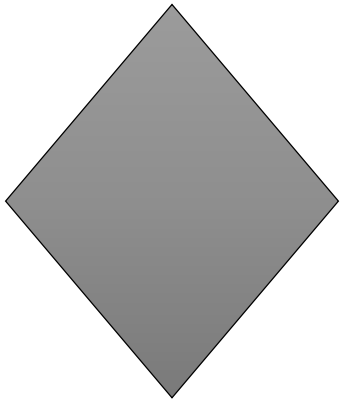
(Alt-Enter shows hints)

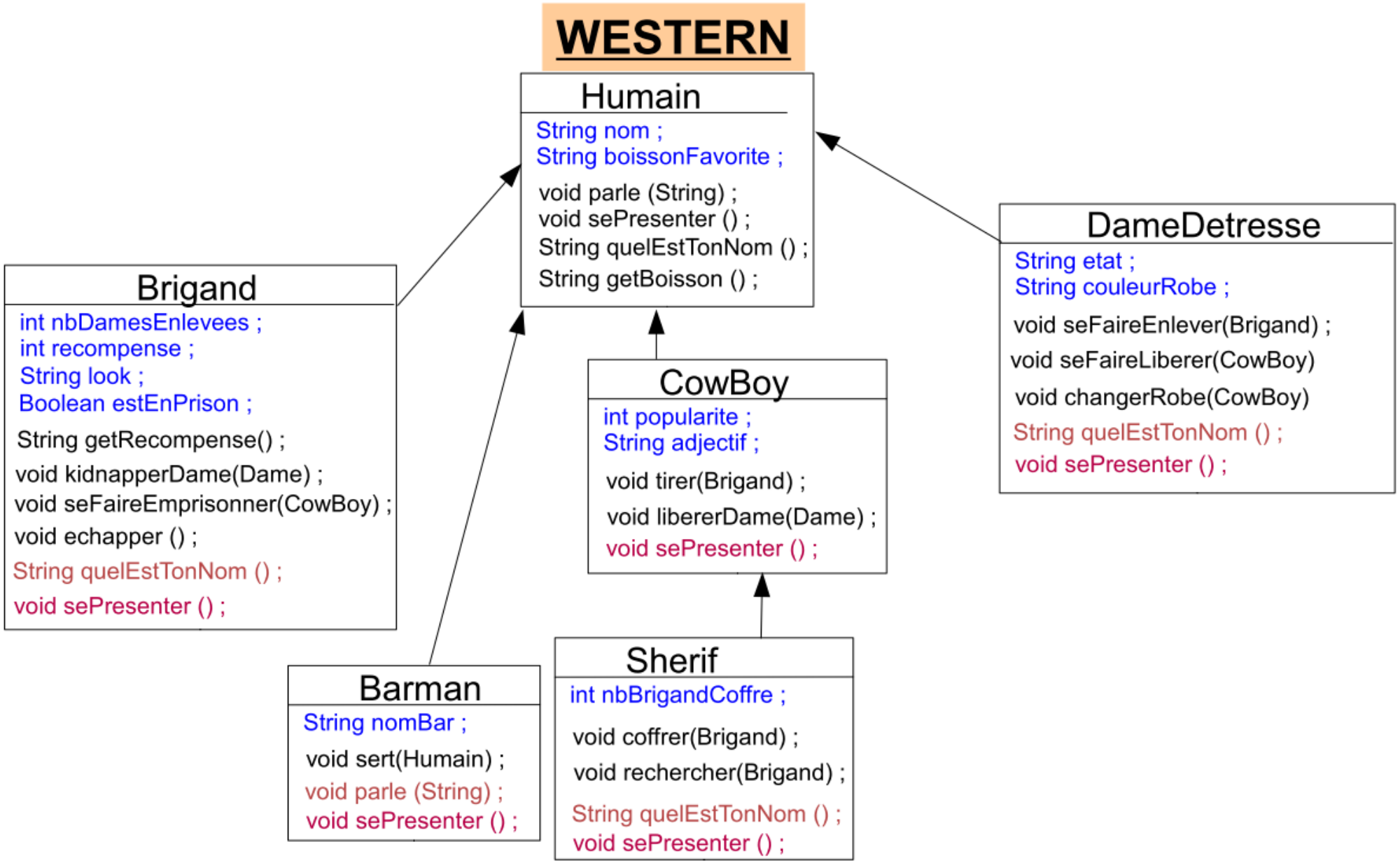
Multiple Inheritance

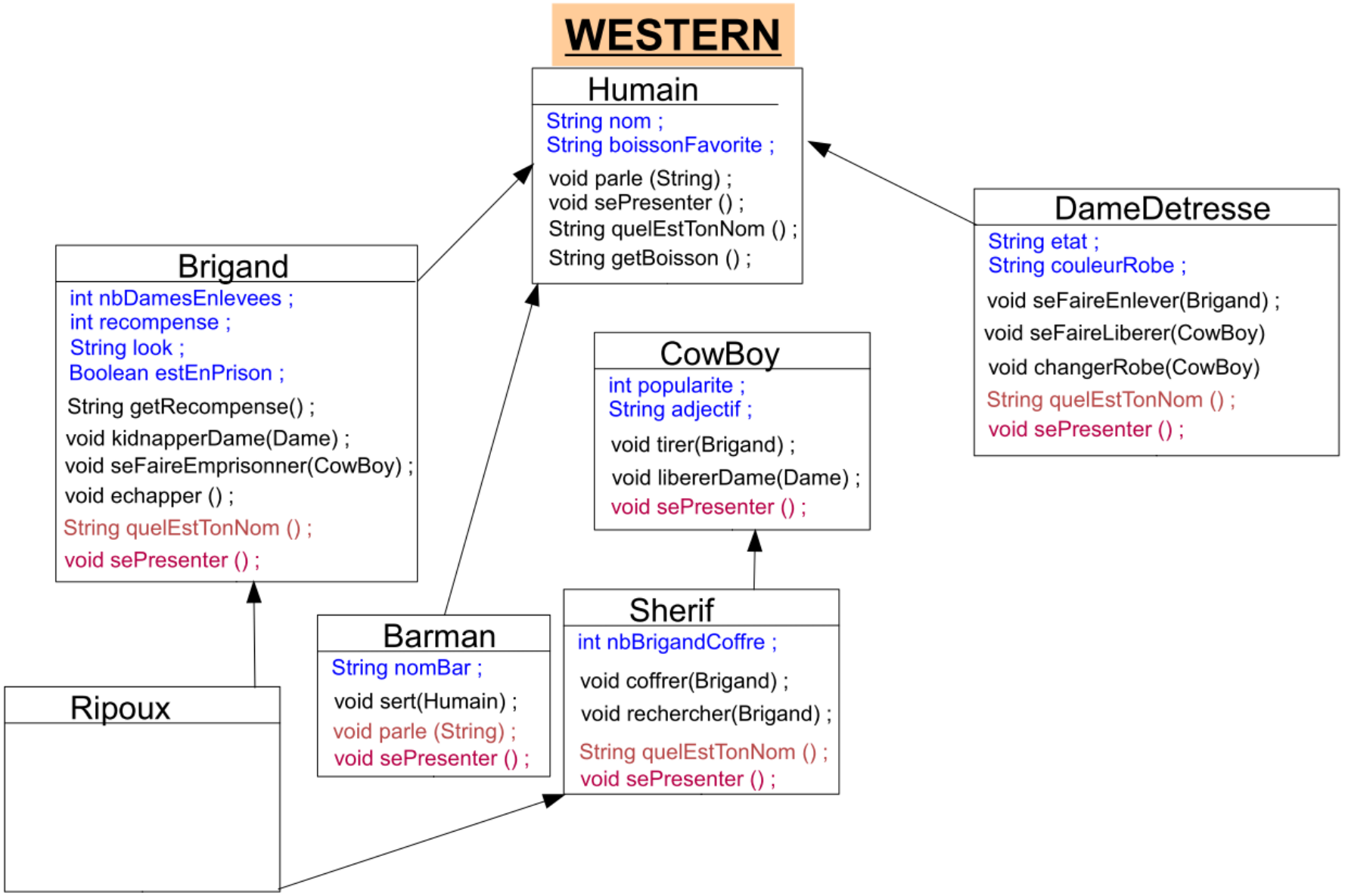
Inheriting from 2 or more classes is Forbidden in Java programming langage

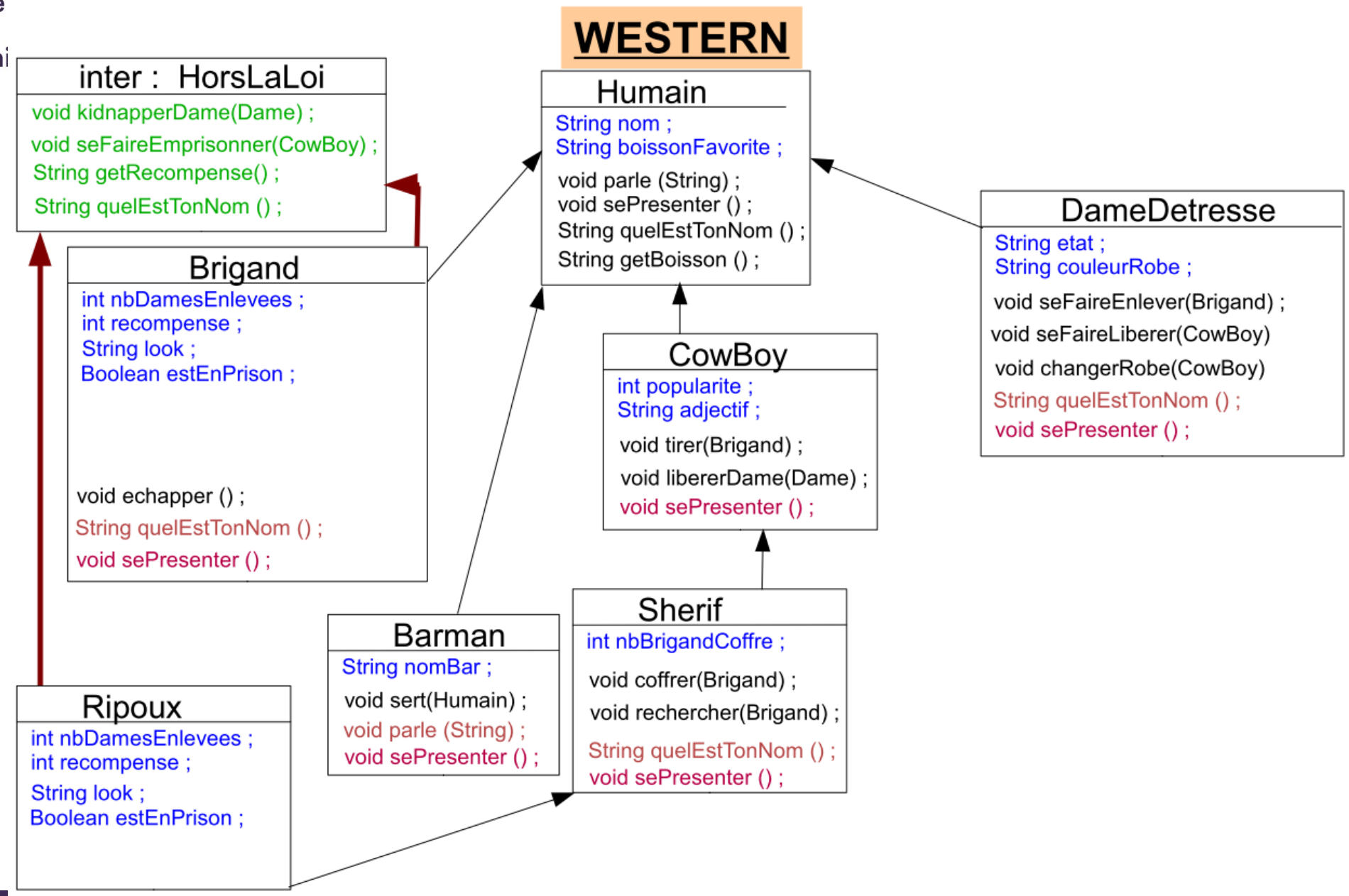
Developers extracted from the multiple inheritance concept :

The idea of a behaviour,
The idea of a contract that a class has to fulfill to meet its requirements




WESTERN
Humain
String nom ;
String boissonFavorite ;
void parle (String) ;
void sePresenter () ;
String quelEstTonNom () ;
String getBoisson () ;
Brigand
int nbDamesEnlevees ;
int recompense ;
String look ;
Boolean estEnPrison ;
String getRecompense() ;
void kidnapperDame(Dame) ;
void seFaireEmprisonner(CowBoy) ;
void echapper () ;
String quelEstTonNom () ;
void sePresenter () ;
CowBoy
int popularite ;
String adjectif ;
void tirer(Brigand) ;
void libererDame(Dame) ;
void sePresenter () ;
DameDetresse
String etat ;
String couleurRobe ;
void seFaireEnlever(Brigand) ;
void seFaireLiberer(CowBoy)
void changerRobe(CowBoy)
String quelEstTonNom () ;
void sePresenter () ;
Barman
String nomBar ;
void sert(Humain) ;
void parle (String) ;
void sePresenter () ;
Sherif
int nbBrigandCoffre ;
void coffrer(Brigand) ;
void rechercher(Brigand) ;
String quelEstTonNom () ;
void sePresenter () ;


WESTERN
Humain
String nom ;
String boissonFavorite ;
void parle (String) ;
void sePresenter () ;
String quelEstTonNom () ;
String getBoisson () ;
Brigand
int nbDamesEnlevees ;
int recompense ;
String look ;
Boolean estEnPrison ;
String getRecompense() ;
void kidnapperDame(Dame) ;
void seFaireEmprisonner(CowBoy) ;
void echapper () ;
String quelEstTonNom () ;
void sePresenter () ;
DameDetresse
String etat ;
String couleurRobe ;
void seFaireEnlever(Brigand) ;
void seFaireLiberer(CowBoy)
void changerRobe(CowBoy)
String quelEstTonNom () ;
void sePresenter () ;
CowBoy
int popularite ;
String adjectif ;
void tirer(Brigand) ;
void libererDame(Dame) ;
void sePresenter () ;
Barman
String nomBar ;
void sert(Humain) ;
void parle (String) ;
void sePresenter () ;
Sherif
int nbBrigandCoffre ;
void coffrer(Brigand) ;
void rechercher(Brigand) ;
String quelEstTonNom () ;
void sePresenter () ;
Ripoux


WESTERN
inter : HorsLaLoi
void kidnapperDame(Dame) ;
void seFaireEmprisonner(CowBoy) ;
String getRecompense() ;
String quelEstTonNom () ;
Humain
String nom ;
String boissonFavorite ;
void parle (String) ;
void sePresenter () ;
String quelEstTonNom () ;
String getBoisson () ;
DameDetresse
String etat ;
String couleurRobe ;
void seFaireEnlever(Brigand) ;
void seFaireLiberer(CowBoy)
void changerRobe(CowBoy)
String quelEstTonNom () ;
void sePresenter () ;
Brigand
int nbDamesEnlevees ;
int recompense ;
String look ;
Boolean estEnPrison ;
void echapper () ;
String quelEstTonNom () ;
void sePresenter () ;
CowBoy
int popularite ;
String adjectif ;
void tirer(Brigand) ;
void libererDame(Dame) ;
void sePresenter () ;
Barman
String nomBar ;
void sert(Humain) ;
void parle (String) ;
void sePresenter () ;
Sherif
int nbBrigandCoffre ;
void coffrer(Brigand) ;
void rechercher(Brigand) ;
String quelEstTonNom () ;
void sePresenter () ;
Ripoux
int nbDamesEnlevees ;
int recompense ;
String look ;
Boolean estEnPrison ;

Interfaces < version 8

- Definition :

  - A list of method signatures (no code)
    - To be implemented in classes
  - A list of constants

  - Methods are implicitly **public abstract**
  - Variables are implicitly **public, static, final**

  - All methods of interface MUST be defined in implemented class

Interfaces < version 8

- Use :
  - Create a reusable architecture
  - Create a new supertype
  - Multiple implementations => More flexible
  - Prefered when no default implementations or state

Interfaces < version 8

- Characteristics:
  - CANNOT instantiate an interface
  - NO constructor
  - ALL methods are abstract
  - NOT extended by a class, is implemented by a class
  - CAN implement multiple interfaces
  - CAN **extend** then **implement** an interface :
    - public class A **extends** B **implements** I1, I2

where: A & B are **classes**

I1 & I2 are **interfaces**

Interfaces > version 8

Drawbacks :

All methods defined in an Interface has to be implemented in classes
Methods in existing Interfaces has to be implemented even if we do not need it (empty methods)
In many situations code redundancy when classes implementing interface use a method to make identical instruction

Solution 1 :

- Possibility to define methods with default body in an interface

```
public interface VisagePale {

    public void aScalp (Indian i);

    default void show(){
        System.out.println("The Indian is angry !!");
    }
}
```

Interfaces > version 8

```java
public interface Interface1 {

    default void show (){
        System.out.println("Interface 1 print method");
    }
}
```

```java
public interface Interface2 {
    default void show (){
        System.out.println("Interface 2 print method");
    }
}
```

```java
public class InterfaceSample implements Interface1, Interface2{

    public static void main(String ... args){
        InterfaceSample inter = new InterfaceSample();
        inter.show();
    }

    @Override
    public void show() {

        Interface1.super.show();
        // OR
        Interface2.super.show();
    }
}
```

```
ut - InterfaceSamples (run)
run:
Interface 1 print method
Interface 2 print method
BUILD SUCCESSFUL (total time: 0 seconds)
```

Interfaces > version 8

Solution 2 :

- Possibility to define static methods with default body in an interface

```
public interface Interface1 {

    public void action();

    static void show (){
        System.out.println("Interface 1 static method");
    }
}
```

Interfaces > version 8

Solution 2 :

Possibility to define static methods with default body in an interface

```
public class InterfaceSample implements Interface1, Interface2{

    public static void main(String ... args){
        InterfaceSample inter = new InterfaceSample();
        inter.action();
        inter.show();
    }

    @Override
    public void show() {
        Interface2.super.show();
    }

    @Override
    public void action() {
        System.out.println("Classical overriding af an abstract method");
    }
}
```

```
ıt - InterfaceSamples (run)
run:
Classical overriding af an abstract method
Interface 2 print method
BUILD SUCCESSFUL (total time: 0 seconds)
```

PART 11

Project

Grading

Final Exam (60%) : Wednesday november 13th (to be verified)

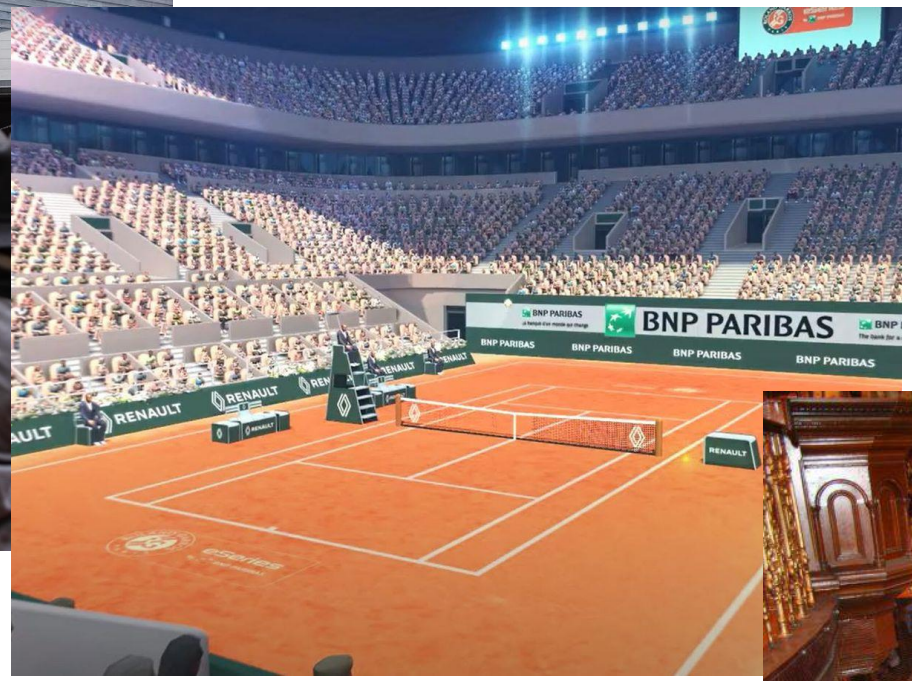
- Understanding of OOP (open course questions, MCQ)
- Short programming exercise,

Mini-Project (40%) :

- Respect of Coding rules is evaluated
- Java Documentation is part of the grid also
- a short report about your project is required (introduction, subject explanation, workthrough samples, difficulties, what make you proud, conclusion)
- Use of every explained Java notions.

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Project Rules



Project Rules

Project subject : Make your own and have fun learning !

Any construction game, multiplayer game are good starts.

However, a program that evaluates the probability of winning a hand in a poker tournament is not suitable for a project here (it is more about an algorithmic complexity than an object oriented project).

* Think about what you want to program
* Make a complete class diagram
* Every java notion seen in class must be part of your project

Due date: Before monday november 11th

mikael.morelle@junia.com

Evaluation Grid

Project				Global			First view			Report	
				Submitted +1, +0.5, 0, -0.5 Too late	Maluses Limit -4.5	Java Standards Limit -2	Java Documentation	Exceptions	User interactions	Intro & Conclusion	Explanations
Lastname	**Firstname**	**Id**	**Projet**				/1 pt	/2 pts	/1 pt	/2 pts	/2 pts
Minimal				-0,50	-4,50	-2,00	0,00	0,00	0,00	0,00	0,00
Maximal				1,00	0,00	0,00	1,00	2,00	1,00	2,00	2,00

Subject		Technical aspects							
Difficulty	Progression	Classes definition, abstract, enum,	Inheritance, Composition	Files	Visibility status	Correct methods	Simple coding, factorized, optimized	Execution	
/1 pt	/1 pt	/2 pts	/1 pt	/1 pt	/1 pt	/2 pts	/1 pt	/2 pts	Assessment
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-7,00
1,00	1,00	2,00	1,00	1,00	1,00	2,00	1,00	2,00	21,00

Maluses for incorrect submission	
Incorrect File name	-1 pt
Incorrect archive format	-1 pt
No directories for source files	-1 pt
Not only the .java files	-1 pt
Teammate not include in submission mail	-0,5 pt

Maluses for incorrect naming	
Constant naming	-0,5 pt
Variables naming	-0,5 pt
Method naming	-0,5 pt
Class naming	-0,5 pt

Submitted date	
7 days or more before	+1 pt
between 2 and 7 days before	+0,5 pt
on time	0 pt
Up to 2 days late	-0,5
more than 2 days late	0/20