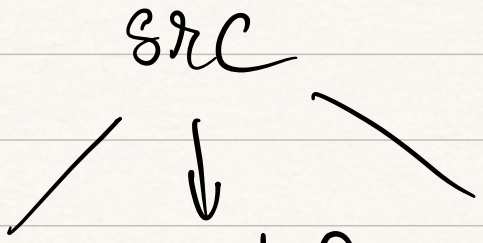


A.java

```
package model.utils;
public class A {
}
```

package (aucun modificateur) → visible partout

```
class A {
}
```

← dans public

← visible seulement dans le package de A

```
public class B {
    void m() {...}
}
```

↑ vu seulement dans le même package

}

package : visible partout
protected : 1) visible partout dans le même package
2) dans un autre package visible seulement par héritage

```
public class A {
  protected void m() {...}
}
class B extends A {
  ...
  this.m(); // légal
  B b = new B();
  b.m(); // légal
}
```

Dans pack'
ailleurs que dans B : (dans class C extends A /B)

```
B b = new B();
b.m(); // erreur sauf si m est redéfinie dans le même package
```

1) On peut étendre la visibilité d'une méthode en la redéfinissant

ex dans B: public void m() {...}

2) On peut en redéfinissant restreindre son type de retour (choisir comme type de retour une référence vers une classe héritière du type initial)

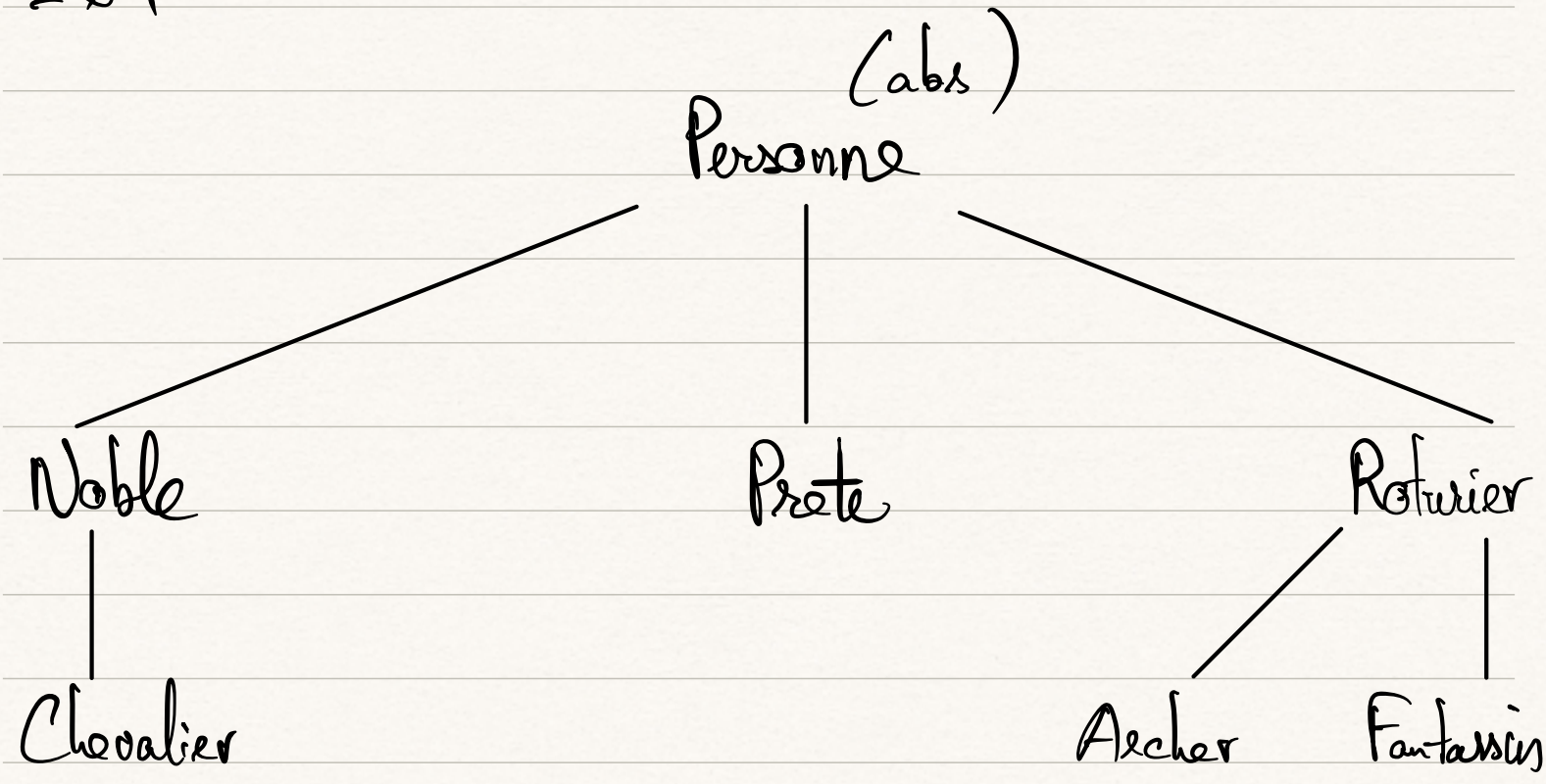
A
|
B

```
class M {
    A m() {...}
}
class N extends M {
    B m() {...}
}
```

TD5

protected = ptd

Ex 1



2. a

Dans Archer

```
void attaque (Personne p){
// p.blessure(p.ptd);  // valide, car dans le meme package
   p.ptd = 0;
}
```

b Dans chevalier

```
private Personne geolier; //null si le chevalier est libre

void attaque (Personne p){
    if(!p instance of Chevalier){
        return;
    }
```

ou

```
    if (p.getClass() != this.getClass()){
        return;
    }
```

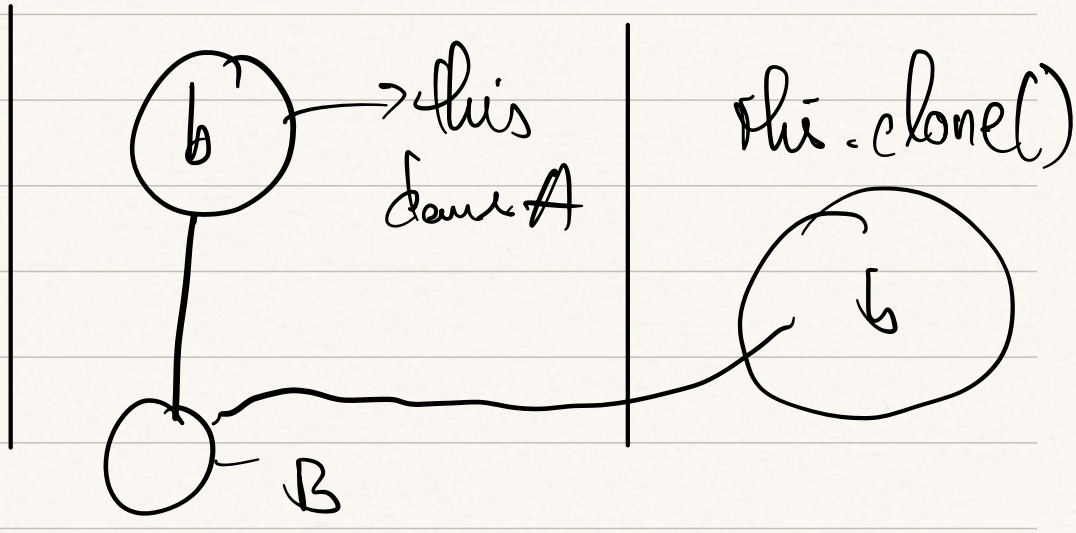
```
Chevalier c = (Chevalier)p;
if (c.geolier != null){
    return;
}
c.geolier = this;
```

cy Dans Fantassin

```
private int degats;
public void attaque (Personne p){
    if (p instanceof Chevalier){
        Chevalier c = (Chevalier)p;
```

```
if (c.geolier != null) {
    return;
}
c.geolier = this; return;
} p.blessure(degats);
}
```

```
class A {
    B b
}
```



Ex 3 : Dans Rotarien extends Personne implement Clonable

Dans Rotarien : re-définir clone

```
public Rotarien clone() throw CloneNotSupportedException
```

Dans Village (class Village implements Clonable)

```
public Village clone() throws ....
```

Ex 4

Dans Rotarier, puis dans Village

```
public boolean equals(Object o){
    if(this.getClass() != o.getClass())
        return false;
    Personne p = (Personne) o;
    return this.nom.equals(p.nom) &&
        this.argent == p.argent &&
        this.pdv == p.pdv
}
```

pdv = point de vie

dans village

```
LinkedList<Rotarier> habitants;
```

```
public boolean equals(Object o){
    if (!o instanceof Village) {
        return false;
    }
    Village v = (village) o;
    if (this.habitants.size() !=
            v.habitants.size()) {
        return false;
    }
}
```

ou

```
int n = habitants.size();
for (int i = 0; i < n; i++) {
    if (!habitants.get(i).equals
            (v.habitants.get(i))) {
        return false;
    }
}
return true;
```

par efficace
1. construire un énumérateur par chaque liste
2. consumer les deux en parallèle

}