Exercice 1 :

class C extends I {} → faux
interface K extends B {} → faux (on ne peut pas aller du concret à l'abstrait)
class C implements J {} → vrai
interface K implements B {} → faux
class extends A implements I {} → vrai
inteface K extends I, J {} → vrai
class C extends A, B {} → faux (il n'y a pas d'héritage multiple en Java)

Exercice 2 :

```
interface Ordre {
	boolean plusGrandQue(Object other) ;
}
interface Ordre <T>{
	boolean plusGrandQue(T other) ;
}  //Méthode propre pour l'exercice, utiliser la généricité

interface Ordre <T extends Ordre<T>>{
	boolean plusGrandQue(Ordre<T> other) ;
}  //Possible mais pas forcément à retenir
```

```
public class ReelG implements OrdreG<Reel> {
	double val;
	public ReelG(double v) { val = v;}

	public boolean plusGrandQue(Reel other){
		return val > other.val;
	}

}
```

```
public class Reel implements Ordre {
	double val;
	public Reel(double v) { val = v;}

	public boolean plusGrandQue(Object other){
		if(other instanceof Double)
			return val > (Double)other;
		else if(other instanceof Reel)
			return val > ((Reel)other).val;
		else
			return false;//throw Exception(); //ou return false ;
		}

}
```

```
public class DiviseurG implements OrdreG<DiviseurG> {
	int val;
	public DiviseurG(int v) { val = v;}

	public boolean plusGrandQue(DiviseurG other){
		return val % other.val == 0;
	}

}
```

```
public class PrefixeG implements OrdreG<PrefixeG> {
	String val;
	public PrefixeG(String v) { val = v;}

	public boolean plusGrandQue(PrefixeG other){
		return val.startsWith(other.val);
	}
```

Exercice 3 :

```java
public interface Chainable {

    Chainable suivant();

    default int longueur(){
        Chainable s = suivant();
        return 1 + ((s == null) ? 0 : s.longueur());
    }
}
```

```java
 * @author Maxime
 */
public class EntierChainable implements Chainable {

    int entier;
    Chainable suivant;

    public EntierChainable(int entier, Chainable suivant) {
        this.entier = entier;
        this.suivant = suivant;
    }

    @Override
    public Chainable suivant() {
        return suivant;
    }

    public String toString(){
        Chainable s = suivant;
        return ""+entier + ((s == null) ?  "" : ", "+s); //On ne met pas toString car Java le sait implicitement
    }
```

```java
/**
 *
 * @author Maxime
 */
public class MotChainable implements Chainable{

    String mot;
    Chainable suivant;

    public MotChainable(String mot, Chainable suivant) {
        this.mot = mot;
        this.suivant = suivant;
    }

    @Override
    public Chainable suivant() {
        return suivant;
    }

    public String toString(){
        Chainable s = suivant;
        return mot + ((s == null) ?  "" : ", "+s); //On ne met pas toString car Java le sait implicitement
    }

}
```

Exercice 5 :

B et X
C et Y