**L’héritage et les interfaces**

L’héritage : extends

L’héritage c’est l’ord ce qu’il une classe hérite d’une autre classe les méthode les même attribues pour pouvoir les utiliser, le mot extends permet de faire l’héritage au langage JAVA.

Une classe fils peut hérité d’une seul classe mère ;

La classe mère par défaut de chaque classe est la classe objet et à la création d’une classe mère la classe objet il ne le sera plus

On peut modifier les méthodes de la classe mère dans la classe fils

public class Voiture extends Vehicule{

}

public class Vehicule {  
 String matiere;  
 void tourner(boolean droite,int anglet){  
   
 System.*out*.println("la voiture est tourné à " + direction + " d'une anglet de : " + anglet);  
 }  
}

dans cette exemple la classe mère hérite la proprieté matiere et la méthode tourner

NB : la classe voiture est la classe fils ; la classe Vehicule est la classe mère

Exercice :

Creation d’une classe fils

public class VesseauDeGuerre extends Vesseau{  
  
 void attaque(Vesseau vesseauattaque,String materierle, int duree){  
 System.*out*.println("un vesseau de type : "+typV+" attque un vesseau de type"+vesseauattaque.typV+"en utilisant l'arme "+materierle+" pendant "+duree+" Minute");  
 vesseauattaque.resistanceDeBuclier=0;  
 vesseauattaque.blindage=vesseauattaque.blindage/2;  
 }  
}

La classe mère

public class Vesseau {  
 String typV;  
 int nbrPassager;  
 int blindage;  
 int resistanceDeBuclier;  
  
 void activerBouclier(){  
 System.*out*.println("le vesseau " +typV+" à activé le bouclier");  
  
 }  
 void desactiverBouclier(){  
  
 System.*out*.println("le vesseau " +typV+" à désactivé le bouclier");  
  
 }  
}

Classe Main

public static void main(String... args){

VesseauDeGuerre chasseur=new VesseauDeGuerre();  
 chasseur.blindage=156;  
 chasseur.resistanceDeBuclier=2;  
 chasseur.typV="CHASSEUR";  
  
 VesseauCivil vesseauMonde=new VesseauCivil();  
 vesseauMonde.blindage=4784;  
 vesseauMonde.resistanceDeBuclier=30;  
 vesseauMonde.typV="Vesseau-Monde";

vesseauMonde.activerBouclier();  
 chasseur.activerBouclier();  
 chasseur.attaque(vesseauMonde,"lasére photomique",3);  
 vesseauMonde.desactiverBouclier();  
  
 System.*out*.println("la résistance d'une bouclier de VM est "+vesseauMonde.resistanceDeBuclier);  
 System.*out*.println("le blindage de VM est "+vesseauMonde.blindage);  
  
}

**Résulta**

le vesseau Vesseau-Monde à activé le bouclier

le vesseau CHASSEUR à activé le bouclier

un vesseau de type : CHASSEUR attque un vesseau de typeVesseau-Mondeen utilisant l'arme lasére photomique pendant 3 Minute

le vesseau Vesseau-Monde à désactivé le bouclier

la résistance d'une bouclier de VM est 0

le blindage de VM est 2392

**Le transtypage**

Qu’on nous voulons qu’un variable soit déclaré comme un type parent

Exmple

Moteur reda = new Voiture();  
reda.cylindre=6;

((Voiture)reda).nom="red";"; ici on a utilité les propriété de la classe fils sans changer le typre de la voiture reda (le transtypage a la volé)

Dans cette exemple on a déclarer une voiture(classe fils) reda de type moteur(classe mère) on peut utiliser seulement le proprietés qui existent dans la classe mère

Il y a la possiblité de changer le type de voiture déclaré de la classe mère au classe fils le cas de besoin d’utilisation des methodes ou attribues de la classe fils, c’es en ajouttant le cede suivant avans d’utiliser les proprieté

Voiture redaVersionVoiture = (Voiture)reda;

**L’heritage et constructeur le mot clé super**

Une classe fille herite les propriété et les methodes d’une classe mère il n’hérite pas les constructeur crées, pour faire cela il faut utiliser le mot clé super

Dans cette exemple on vas crée un constructeur dans la classe véhicule(mère) ce constructeur vas rappeler le constructeur de la classe moteur pou pouvoir l’utiliser dans la classe voiture

Classe mère

public class VehiculeAmoteur {  
  
 Moteur moteur ;  
 int vitesse ;  
  
 VehiculeAmoteur(){  
  
 }  
 VehiculeAmoteur(Moteur moteur){  
 this.moteur=moteur;  
 System.*out*.println("un voiture est construite avec le moteur ");  
 }  
  
}

Classe fille

public class Voiture extends VehiculeAmoteur{

Voiture(){  
 super();  
}

Ou

Voiture(){  
 super(new Moteur()); dans ce cas il faut ajouter super(new Moteur()); ou this ; dans tous les constructeur de la classe voiture Voiture

}

/////////////////////////////////////////////////  
Voiture(Moteur moteur){  
 super(moteur);  
}  
public Voiture() {  
  
}

Dans la classe main pour la creation avec ce consructeur de la classe mère

Moteur moteur1= new Moteur();

Voiture reda1 = new Voiture(moteur1);

**Covariance des methodes**

Cl creation d’une classe UsineDassemblage dans laquel on declare une methode qui intialise VehiculeAmoteur

public class UsineDassemblage {  
  
 VehiculeAmoteur assemblageM(){  
 Moteur moteur=new Moteur();  
 VehiculeAmoteur vam=new VehiculeAmoteur(moteur);  
 return vam;  
 }  
  
}

creation d’une classe qui hérite de la classe UsineDassemblage et redifinir la methode assemblageM

public class UsineDassemblageVoiture extends UsineDassemblage{  
  
 Voiture assemblageM(){  
 Voiture v =new Voiture();  
 return v;  
 }  
}

l’utlisation de la methode assemblageM avans et après la redifinition (la modification du type de retour )

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

**avans la redifinition**

UsineDassemblage ua=new UsineDassemblage();  
 VehiculeAmoteur vam=ua.assemblageM();

**Après la redifinition**

UsineDassemblageVoiture u=new UsineDassemblageVoiture();  
 Voiture v=u.assemblageM();  
  
 }  
}

**L’interface : le mot clé implement**

H il sagit d’un héritage multiple c’est l’utilisation des interfaces

Une classe peut hériter une seul classe et implementer plusieur interface

Une interface peut hériter une autre interface

Exemple

Ici on va crée une interface videnger qu’on peut utiliser dans plusieur classe ca depent le besoin (on vas faire le videnge pour une voiture ou pour un lavabo … )

Interface avec methode

public interface Amarrable {  
  
// methode Nombre de corde  
 public int combienDecorde(int vitesseduvent);  
}

**Classe bateau utilse l’interface et redifinie la methode**

public class Bateau implements Amarrable{  
 int masse;  
 @Override  
 public int combienDecorde(int vitesseduvent) {  
 int nbcord=masse/10;  
 nbcord=nbcord+vitesseduvent/100;  
 return nbcord;  
 }  
}

public class origin {  
 public static void main(String[] args){  
  
 Bateau b=new Bateau();  
 b.masse=60;

Port p=new Port();  
 p.acceuilleEngin(b);  
  
 }  
}

**Resulta**

le nombre de corde necessaire est : 6

NB : une interface peut hériter un ou plusieurs interfaces

public interface Amarrable extends Videngeable,interface2{

Interface Drappeau et serializable : Des interfaces vide fondamentale ??

**NB** : une interface peut avoir des propriétés avec des valeurs fix

**Le Mot clé instanceof**

Permet de tester la nature d’un objet crée

public class Port {  
  
 void acceuilleEngin(Amarrable amarable){  
 if (amarable instanceof Bateau){  
 System.*out*.println("c'ette amarable est un bateau");  
 }  
 int nbcorde=amarable.combienDecorde(50);  
 System.*out*.println("le nombre de corde necessaire est : "+nbcorde);  
 }  
}

dans cette exemple on a tester si le amarable crée est un bateau on ecrit le message de printLn

**Exemple exercice video 81**

public class PlaneteTellurique extends Planete implements Habitable{  
  
  
 public PlaneteTellurique(String nom){  
 super(nom);  
 }  
  
 public Vesseau acceuillirVesseau(Vesseau vesseau) {  
 if (vesseau instanceof VesseauDeGuerre){  
 ((VesseauDeGuerre)vesseau).desactiverArme();  
 }  
 nbrH = nbrH + vesseau.nbrPassager;  
 vesseauA = vesseau;  
 if (vesseauA == null) {  
 System.*out*.println("aucun vesseau ne s'en va");  
 }  
 else {  
 System.*out*.println("un vesseau de type :" + vesseauA.typV + " doit s'en aller");  
 }  
 return vesseau;  
  
 }  
}

public class VesseauDeGuerre extends Vesseau{  
 boolean armedesactiver;  
 void attaque(Vesseau vesseauattaque,String materierle, int duree){  
 if (armedesactiver){  
 System.*out*.println("l'attaque impossible arme désactiver");  
 }  
 System.*out*.println("un vesseau de type : "+typV+" attque un vesseau de type"+vesseauattaque.typV+"en utilisant l'arme "+materierle+" pendant "+duree+" Minute");  
 vesseauattaque.resistanceDeBuclier=0;  
 vesseauattaque.blindage=vesseauattaque.blindage/2;  
 }  
 void desactiverArme(){  
 armedesactiver=true;  
 System.*out*.println("l'arme est désactivé "+typV);  
 }  
}

le polymorphisme(de sous-typage)

permet de considerer un objet comme un autre compatible

une methode peut accepte un objet sous la forme de son type parent

Exercice Video 83 a refaire

@Override  
 void activerBouclier(){  
 this.desactiverArme();  
 super.activerBouclier();  
 System.*out*.println("le vesseau " +typV+" à activé le bouclier");  
  
 }  
}

Main

Vesseau chasseur=new VesseauDeGuerre();  
chasseur.blindage=156;  
chasseur.resistanceDeBuclier=2;  
chasseur.typV="CHASSEUR";

chasseur.activerBouclier();  
((VesseauDeGuerre) chasseur).attaque(vesseauMonde,"lasére photomique",3);  
vesseauMonde.desactiverBouclier();

**Methode et classe abstraite(abstract)**

abstraite est une classe qui ne peut pas etre instyancier, il a le but de generaliser d’autre classes

la classe abstraite prut avoir de propriétés des méthodes aussi des méthode spéciale que l’on appel abstrait et aui n’on pas un bloc d’instruction

exemple classe abstract

public abstract class Vehicule {  
 abstract void klaxoner();  
}

une classe abtract peut etre hérité par une autre classe et de redifinir les méthodes déclarés ça dépend le besoin ça donne plus de flixibilité à lutilisation des méthodes

public class Voiture extends Vehicule implements Videngeable {

@Override  
 void klaxoner() {  
 System.*out*.println("tuuutuuu!!!!!");  
 }  
}

public class Moteur extends Vehicule implements Videngeable {

@Override  
 void klaxoner() {  
 System.*out*.println("TooooooTOOOOOOOOOOOOPPOOOOOOO!!!!!");  
  
 }  
}

**NB** : si on veut redéfinir les méthodes dans les classe sans crée une classe abstract il faut les rendre aussi abstract, l’inconvénient c’est qu’on peut pas instancier ces classe(crée des objets)

Exercice a refaire