Héritage : les champs private de la mère ne sont pas accessible directement par la fille, il faut un getter ou super.champs

Le main est toujours static car cela permet son invocation automatiquement à l'exécution sous création d'un objet de la classe test.

une meth static peut accéder uniquement a des membres static de sa class et n'a pas accès à this.

this doit être le premier dans le constructeur s'il est présent.

Meth d'instance peut accéder à tous les membres de sa classe (static ou non)

Surcharge : même type de retour, même nom et des paramètres différents.

Redéfinition : même signature et corps différent. Le type de retour peut changer mais il doit être sous type du 1er.

On peut changer les modificateurs d'accès lors de la redéfinition mais pour élargir : Private<package< protected< public.

meth final ne peut pas être redéfinie.

L'initialisation se fait par 3 parties : Par défaut : tout à zéro et à null. | Bloc initialiseurs. | -Corps du constructeur.

Type getP(return p ;) -> violation d'encapsulation car p est une référence, du coup on pourra le modifier dans une autre classe.

get class retourne le type effectif.

S'il y a redéfinition des meth, c'est le type effectif qui détermine les meth à utiliser.

Avec les méthodes static et les variables ca ne change pas, ceux du déclaré qui sont utilisés.

- passage de paramètres :
- les changements sur des paramètres de type primitif n'affectent pas les variables du main.
- les changements sur les paramètres reference affecte les variables du main.
- les changements sur les références n'affectent pas le main.

Classe Membre : un objet de la classe membre peut être créé uniquement par un objet de la CE en utilisant juste new ClassMem()

L'objet de la CE est accessible par l'objet de la classe interne : ClassEnglo.this

- une classe membre non static ne peut avoir des méthodes/champs static.
- la classe membre a accès aux champs de la CE même s'ils sont private ...
- l'englobante a accès au champ/méthodes de sa classe membre même si private.
- si une classe membre est visible hors sa CE elle est dénotée ClassEnglo.ClassMem

Public class F{ public class H{..}}

Main; F a= new F(); F.H=a.new H();

A condition que H soit public.

- classe membre static a accès uniquement aux champs static de la CE.

main : public class A{ public static class B{..}}

A.B nom = new A.B(); (condition B soit public)

- classe static a accès aux champs non static des classes membres private de la CE.
- pour accéder à un champ de la CE on utilise ClassEng.this.champs
- CL définie dans un bloc de code
- elle a accès aux champs de la CE et le champs du bloc englobant (ceux du bloc doivent être (effectively) final, elle a accès aux param d'une méthode mais elle ne peut pas initialiser les champs/val)

Up ?action1 :action2 -> si up est true alors on fait action1 sinon action2.

- classe anonyme est une CL sans nom.
- créer un objet de classe anonyme qui hérite une classe de base : NomClassBase nom= new Class Base(paramètre de base){Champs de plus et méthodes de plus}
- elle n'a pas de constructeur.
- this dans l'expression lambda fait référence à l'objet de la CE et non l'objet de la classe anonyme créée.
- expression lambda pour inverser un tableau

TabOperation t= (int[] t)->{ inti nv[] =new int[t.length]; for(i=0;i<length,i++){inv[i] = tab[length-i-1]; } return inv}

Somme: fonctionnelle< Integer> addition = (a, b) -> a + b;

Les classes abstraites contiennent au moins 1 meth abstraite-> on peut pas créer des objets de cette classe.

Les interfaces n'ont que des champs constants et la plupart des méthodes sont abstraites et sont automatiquement public –> on déclare des objets de type interface mais on les instancie pas.

on peut avoir des méthodes static qui doivent avoir du corps, et aussi des méthodes private qui seront utilisées par les autres méthodes de l'interface. Les 2 sont pas accessibles par les classes qui implémentent l'interface

il y a des méthodes default, les classes qui implémentent l'interface peuvent ne pas les redéfinir.

B extends A implements I1 : I1 contient f() et A aussi -> B contiendra celle de A(sa mère)

B implements i1, I2 -> I1 contient f() (définie) et I2 contient f() (non définie) -> B doit redéfinir f()

une interface peut étendre 2 interfaces.

Methode générique peut appartenir a une classe générique et non generique.

On utilise extends pour la bornes des classes et interfaces à la fois. Tous les types extends Object.

T peut être borné par plusieurs interfaces mais par 1 seule class(on mentionne la classe en 1er)

Si A est un sous type de B alors c<A> N'EST PAS un sous type de c<B>. exemple : Student extends Personne mais arrayList<Student> n'est pas sous type de arrayList<Personne>.

Par contre A[] t1 est sous type de B[] t2

? est utilisé quand aucun autre type ne dépend de lui, car il ne doit pas être déclaré

? peut être aussi borné mais par 1 seule borne :

? extends machin, ? super machin

si B extends A : c<B> extends c< ?extends A> et

C<A> extends c< ?super B>

Il ya deux types d'exceptions :

- checked : héritent de Exception
- Unchecked : héritent de RunTimeException

On utilise throws devant les meth qui soulèvent une exception checked.

Si la meth la capture avec try ... catch pas besoin de throws.

Si une exception est capturée avec try catch (sans qu'on fasse throw a l'intérieur du bloc catch) le programme continuera l'exécution.

s'il y a throw a l'intérieur du bloc catch on exécute le finally puis on soulève l'exception.

Si on return au bloc catch on exécute d'abord le finally puis on retourne.

si aucun catch ne correspond a l'exception on exécute le finally puis on affiche thread main ...

les blocs catch doivent être classés du plus spécifique au plus général.

swing est développé en utilisant le design pattern MVC : model view controller.

Il existe 4 concept en poo : héritage, encapsulation, polymorphisme et abstraction

Td6 et 8 : On ne peut instancier ni une interface ni une classe abstraite.

- les classes abstraites ont des constructeurs, mais non pas les interfaces.
- A a = new B() OK -> si A est abstract, B non Abstract et A extends B
- A a = new B() : si A est une interface et B implements A OK
- Interface ont des attributs constants(public static final) et doivent être initialisés. Les classes abstraites c'est comme les classes, pas de restriction
- on peut avoir des méthodes static abstract dans les classes abstraites. Dans les interfaces on peut avoir des abstract/ des static mais pas static abstract.
- Il faut catch d'abord les classes filles puis les classes mères sinon erreur à l'exécution.
- Finally s'exécute même s'il y a erreur

---

public class A {} public class B extends A {}

toutes les méthodes de la classe A sont héritées par la classe B, → la classe B peut déclarer une méthode avec la même signature qu'une méthode de la classe A

XXXX squaredLength = YYYY ; String s = " hello "; System.out.println (squaredLength.applyAsInt (s));

ToIntFunction<String> squaredLength = s -> s.length()*s.length();

---

public class A {} public class B extends A {}

ArrayList<B> est un ST de ArrayList<? extends B>

× ArrayList<A> est un ST de ArrayList<? super B>,

× ArrayList<A> est un ST de ArrayList<B>,

ArrayList<B> est un ST de ArrayList<? super A>

Lorsqu'une classe A étend une classe B :

→ toutes les instances de A sont aussi des instances de B

× toutes les instances de B sont aussi des instances de A

× les deux réponses précédentes sont vraies

× les trois réponses précédentes sont fausses.

---

public class Voiture { public int kmParcourus ; } public class Ferrari extends Voiture { }

x Pour pouvoir créer des voitures et des Ferraris neuves (ont parcouru 0 kilomètre) il faut ajouter un constructeur dans les deux classes.

× On peut ajouter un constructeur dans Voiture qui construit des voitures avec un kilométrage non nul, sans ajouter de constructeur dans Ferrari.

→ On peut ajouter un constructeur dans Ferrari qui construit des Ferrari avec un kilométrage non nul, sans ajouter de constructeur dans Voiture.

× On peut ajouter un constructeur dans les deux classes, uniquement sous réserve que les deux aient les mêmes arguments.
