|  |
| --- |
| **INF 1220** |
| **Introduction à la programmation** |

|  |
| --- |
| **Travail noté 5 - Héritage et polymorphisme** |
| 20 points |

|  |  |
| --- | --- |
| NOm : Nichols | PRÉNOM : Martin |
| numÉro d’Étudiant : 20282501 | TRIMESTRE : Hiver 2025 |
| Adresse : 764 Rue Principale, La Présentation | |
| CODE POSTAL : J0H 1B0 | TÉLÉPHONE DOMICILE : (450) 796-5054 |
| TÉLÉPHONE TRAVAIL | CELLULAIRE : (450) 223-4018 |
| Courriel : nichols.martin@univ.teluq.ca  NOM DE LA PERSONNE TUTRICE : Khadidja Henni | |
| Date D’ENVOI : 21 septembre 2025 | |
| **Réservé à l’usage de la personne tutrice** | |
| Date DE RÉCEPTION | Date DE RETOUR |
| NOTE | |

Une image contenant Police, texte, Graphique, logo

Le contenu généré par l’IA peut être incorrect.

Table des matières

[Question #1 3](#_Toc209271190)

[Question #2 4](#_Toc209271191)

[TestCercle.java 4](#_Toc209271192)

[Carre.java 5](#_Toc209271193)

[Cercle.java 5](#_Toc209271194)

[Resizable.java 6](#_Toc209271195)

[Forme.java 6](#_Toc209271196)

[Explications 6](#_Toc209271197)

[Question 3 7](#_Toc209271198)

[Réponse : 7](#_Toc209271199)

[Bibliographie 8](#_Toc209271200)

# Question #1

Veuillez expliquer en quelques phrases les résultats ou les erreurs lors de l'utilisation des classes dans la classe TestAnimaux.java:

L’exécution des codes fournis sur le site du cours donnent l’erreur suivante :

|  |
| --- |
| Exception in thread "main" java.lang.ClassCastException: class Chien cannot be cast to class Chat (Chien and Chat are in unnamed module of loader 'app')  at TestAnimaux.main(TestAnimaux.java:25) |

Au début tout se passe correctement. On instancie chaque classe Chat, Chien et GrosChien et on appelle leur implantation de greeting(). Les trois premières sorties sont correctes :

|  |
| --- |
| Meow!  Woof!  Woow! |

Dans la seconde partie du code on utilise le polymorphisme pour créer deux variables contenant un Chien et une variable contenant un GrosChien en utilisant une référence à la classe abstraite Animaux. Dans cet extrait du code, Animaux animal1 = new Chien(); , on commence par une référence au type abstrait Animaux et on y crée un objet Chien. En faisant animal1.greeting(); on appelle donc le greeting() de l’animal créé à travers la référence de type Animaux et le type de l’objet, Chien ou GrosChien, détermine ce qui sera affiché en console. La sortie est bien celle attendue :

|  |
| --- |
| Woof!  Woof!  Woow! |

Dans la dernière partie du main on retype les références en Chien ou GrosChien pour appeler la méthode surchargée greeting(Chien). La sortie en console est celle attendue jusqu’au cast final Chat cat2 = (Chat)animal2; . dans ce dernier cast on tente de convertir un chien en chat. C’est cette dernière ligne qui lève une exception. En créant animal2 plus haut dans le code il a été spécifié que c’est un objet Chien. Ce dernier ne peut pas être transformé en Chat par la suite.

# Question #2

Vous devez appliquer l'héritage afin que Cercle et Carré héritent à la fois de Forme et Resizable. Vous pouvez bien entendu modifier le code reçu. Utilisez votre bon jugement. Votre solution doit comprendre au moins une classe abstraite. Vous devez expliquer vos choix de manière détaillée: un travail remis avec des explications insuffisantes pourra se voir attribué la note de zéro, sans droit de reprise.

## TestCercle.java

|  |
| --- |
| public class TestCercle {      public static void main(String[] args) {          System.out.println("Bonjour, INF 1220 !");          Cercle c = new Cercle(1.9);          c.resize(3.0);          System.out.println("Après resize: " + c);          Carre s = new Carre(2.0);          System.out.println(s);          s.resize(5.0);          System.out.println("Après resize: " + s);          Forme f1 = c;          Forme f2 = s;          System.out.println("Périmètres via Forme: " + f1.getPerimeter() + " ; " + f2.getPerimeter());          // Démo « double héritage » (classe + interface)          Forme[] formes = { new Cercle(2), new Carre(1) };          for (Forme f : formes) {              System.out.println(f.getClass().getSimpleName() + " : " + f.getPerimeter());              ((Resizable) f).resize(10); // possible car chaque Forme ici est aussi Resizable              System.out.println("Après resize: " + f.getPerimeter());          }      }  } |

## Carre.java

|  |
| --- |
| public class Carre extends Forme implements Resizable {      private double taille;      public Carre(double taille) {          setTaille(taille);      }      public double getTaille() {          return taille;      }      @Override      public double getPerimeter() {          return 4 \* taille;      }      @Override      public void resize(double taille) {          setTaille(taille);      }      private void setTaille(double t) {          if (t <= 0) throw new IllegalArgumentException("La taille doit être > 0");          this.taille = t;      }  } |

## Cercle.java

|  |
| --- |
| public class Cercle extends Forme implements Resizable {      private double diametre;      public Cercle(double diametre) {          setDiametre(diametre);      }      public double getDiametre() {          return diametre;      }      @Override      public double getPerimeter() {          return Math.PI \* diametre;      }      @Override      public void resize(double diametre) {          setDiametre(diametre);      }      private void setDiametre(double d) {          if (d <= 0) throw new IllegalArgumentException("Le diamètre doit être > 0");          this.diametre = d;      }  } |

## Resizable.java

|  |
| --- |
| public interface Resizable {      void resize(double d);  } |

## Forme.java

|  |
| --- |
| public abstract class Forme {      public abstract double getPerimeter();  } |

## Explications

* Resizable est transformé en interface. Cercle et Carre doivent hériter tous deux de Forme et Resizable et la seule manière trouvée pour respecter toutes les conditions de l’énoncé de la question (conserver au moins une classe abstraite) est cette transformation. Les classes concrètes pourront donc faire extends Forme et implements Resizable.
* Forme reste tel quel. Aucun changement n’y est nécessaire.
* Cercle.java et Carre.java font maintenant extends Forme implements Resizable.
* Forme contient un contrat commun : getPerimeter et ne connais pas les autres détails. Il serait possible d’y ajouter d’autres propriétés communes si désiré.
* Les dimensions sont privées et toujours validées. Leur utilisation se limite aux classes dans lesquelles on les trouve et on évite des valeurs invalides en s’assurant qu’elles sont toutes >0.
* @Override sert à sécuriser la redéfinition.
* Il est possible de travailler sur un Cercle ou un Carre avec une référence de type Forme et appeler getPermieter().
* Toutes les formes, même une nouvelle, peut utiliser Resizable pour être redimensionnée.
* IllegalArgumentException sert à clarifier les erreurs d’usage dans les cas où la valeur serait t <= 0.

# Question 3

À quoi servent les interfaces en Java ? Utilisez la question précédente (2) comme base de discussion. Maximum 5 phrases.

## Réponse :

L’interface est « un contrat qui définit un ensemble de méthodes abstraites... qu’une classe doit implémenter » (*L’héritage, les classes abstraites et les interfaces*, s. d.). On peut utiliser plusieurs interfaces dans une classe pour définir des comportements standardisés. Dans la solution présentée à la question précédente, on a fait de Resizable une interface, ce qui permet à n’importe quelle classe de l’implémenter. On peut ensuite manipuler les objets via Resizable sans connaitre leur type concret. Le comportement et la forme sont donc ainsi séparés, ce qui permet d’ajouter une nouvelle forme redimensionnable sans avoir à modifier le code.

# Bibliographie

*L’héritage, les classes abstraites et les interfaces*. (s. d.). INF 1220 - Introduction à la programmation. Consulté 15 septembre 2025, à l’adresse https://lemire.github.io/inf1220-hugo/docs/modules/module5/activite-5-1/