JAVA - Exceptions, Sets, Threads & Files

Exception Handling

Une exception est un problème qui se produit lors de l'exécution du programme. Les exceptions entraînent un arrêt anormal du programme.
La gestion des exceptions est un mécanisme puissant qui gère les erreurs d'exécution pour maintenir le flux d'applications normal.

Une exception peut se produire pour plusieurs raisons différentes. Quelques exemples:
- Un utilisateur a saisi des données non valides.
- Un fichier qui doit être ouvert ne peut pas être trouvé.
- Une connexion réseau a été perdue au milieu des communications.
- Mémoire insuffisante et autres problèmes liés aux ressources physiques.

Comme vous pouvez le constater, les exceptions sont causées par une erreur de l'utilisateur, une erreur du programmeur ou des problèmes de ressources physiques. Toutefois, un programme bien rédigé devrait traiter toutes les exceptions possibles.

Une déclaration catch implique de déclarer le type d'exception que vous essayez de capturer. Si une exception se produit dans le bloc try, le bloc catch qui suit l'essai est caché. Si le type d'exception qui est survenu est répertorié dans un bloc catch, l'exception est passée au bloc catch beaucoup comme un argument est passé dans un paramètre de méthode.
Le type Exception peut être utilisé pour prendre toutes les exceptions possibles.

L'exemple ci-après illustre la gestion des exceptions lors de l'accès à un index de tableau qui n'existe pas:

```
Java
1  public class MyClass {
2      public static void main(String[ ] args) {
3          try {
4              int[ ] a = new int[2];
5              System.out.println(a[3]);
6          } catch (Exception e) {
7              System.out.println("An error occurred");
8          }
9      }
10 }
```

//Outputs "An error occurred"

Syntax:

```
try {
//some code
} catch (Exception e) {
//some code to handle errors
}
```

Gestion des exceptions

Les exceptions peuvent être prises en utilisant une combinaison des mots-clés try et catch.
Un bloc try / catch est placé autour du code qui peut générer une exception.

Sans le bloc try / catch, ce code devrait bloquer le programme, comme a(3) n'existe pas.
Notez l'instruction (Exception e) dans le bloc catch– elle est utilisée pour capturer toutes les Exceptions possibles.

```
Fill in the blanks to handle any possible exceptions.

int x = 12;
int y = 0;
int z;
try {
  z = x / y;
  System.out.println(z);
}
catch (Exception e) {
  System.out.println("Error");
}
```

Multiple Exceptions

Fill in the blanks to create a method that throws an IOException if the parameter is negative.

```
public void do(int x)
  throws IOException {
  if(x<0)
  {
    throw new IOException();
  }
}
```

L'instruction throws dans la définition de la méthode définit le type d'exception (s) que la méthode peut lancer.
Ensuite, le mot clé throw jette l'exception correspondante, ainsi qu'un message personnalisé.
Il nous appelons la méthode dire avec le second paramètre égal à 0, il lancera un ArithmeticException avec le message "Division par zéro".
Plusieurs exceptions peuvent être définies dans l'instruction throws en utilisant une liste séparée par des virgules.

```
Java
1  import java.io.*;
2
3  class Program {
4      static void do(int x, int b) throws ArithmeticException {
5          try {
6              int[] a = new int[2];
7              a[0] = 1 / b;
8              if (x < 0)
9              {
10             throw new IOException("Division by zero");
11             }
12             else a[1] = 5;
13         }
14         catch (ArithmeticException e) {
15             System.out.println("Division by zero");
16         }
17     }
18     public static void Main(string[] args) {
19         do(-1, 0);
20     }
21 }
```

```
Exception in thread "main"
java.lang.ArithmeticException:
    Division by Zero at
    Program.div(Program.java:8) at
    Program.main(Program.java:13)
```

jetor

Le mot-clé throw vous permet de générer manuellement des exceptions à partir de vos méthodes.
Certains des nombres types d'exception disponibles incluent l'exception IndexOutOfBoundsException, IllegalArgumentException, ArithmeticException et ainsi de suite.
Par exemple, nous pouvons lancer une ArithmeticException dans notre méthode lorsque le paramètre est 6.

Gestion des exceptions

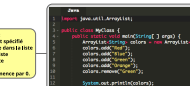
Un bloc d'essai simple peut contenir plusieurs blocs de capture qui gèrent différentes exceptions séparément.

Tous les blocs de capture doivent être commandés de la plus spécifique à la plus générale.
À la suite des exceptions spécifiques, vous pouvez utiliser le type Exception pour traiter toutes les autres exceptions comme dernière capture.

```
try {
  //some code
} catch (ExceptionType1 e1) {
  //Catch block
} catch (ExceptionType2 e2) {
  //Catch block
} catch (ExceptionType3 e3) {
  //Catch block
}
```

Threads

"Multi-File" (Threads), java est un langage de programmation multi-thread. Cela signifie que notre programme peut utiliser au mieux les ressources disponibles en exécutant simultanément deux composants ou plus, chaque composant traitant une tâche différente.
Vous pouvez subdiviser des opérations spécifiques au sein d'une même application en threads individuels qui fonctionnent tous en parallèle.
Le diagramme suivant montre le cycle de vie d'un thread.



Il existe deux façons de créer un thread.
1. Étendre la classe Thread
Hériter de la classe Thread, remplacer sa méthode run () et écrire la fonctionnalité du thread dans la méthode run().
Ensuite, vous créez un nouvel objet de votre classe et appelez sa méthode de démarrage pour exécuter le thread.

```
Java
1  class Loader extends Thread {
2      public void run() {
3          System.out.println("Hello");
4      }
5  }
6
7  class MyClass {
8      public static void main(String[ ] args) {
9          Loader obj = new Loader();
10         obj.start();
11     }
12 }
```

Comme vous pouvez le voir, notre classe Loader étend la classe Thread et remplace sa méthode run ().
Lorsque nous créons l'objet obj et appelons sa méthode start (), les instructions de la méthode run () s'exécutent sur un thread différent.

Chaque thread java est hiérarchisé pour aider le système d'exploitation à déterminer l'ordre de planification des threads. Les priorités varient de 1 à 10, chaque thread devant passer en priorité 5. Vous pouvez définir la priorité de thread avec la méthode setPriority ().

L'autre façon de créer Threads est de mettre en œuvre l'interface Runnable.
Implémenter la méthode run (), ensuite, créer un nouvel objet Thread, passez la classe Runnable à son constructeur et lancez le Thread en appelant la méthode start ().

```
Java
1  class Loader implements Runnable {
2      public void run() {
3          System.out.println("Hello");
4      }
5  }
6
7  class MyClass {
8      public static void main(String[ ] args) {
9          Thread t = new Thread(new Loader());
10         t.start();
11     }
12 }
```

La méthode Thread.sleep () met en pause un Thread pour une période de temps spécifiée. Par exemple, appeler Thread.sleep (1000); Met le fil en pause pendant une seconde. Gardez à l'esprit que Thread.sleep () jette une InterruptedException, alors assurez-vous de l'entourer d'un bloc try / catch.

Il peut sembler que la mise en œuvre de l'interface Runnable est un peu plus compliqué que l'extension de la classe Thread. Toutefois, la mise en œuvre de l'interface Runnable est la meilleure façon de démarrer un Thread, car elle vous permet également d'étendre à partir d'une autre classe.

```
class A implements Runnable {
  public void run(){
  System.out.println("Bye");
  }
}
public class App {
  public static void main(String[ args) {
    Thread ob = new Thread(new A, "A");
    ob.start();
  }
}
```

Remplissez les espaces pour mettre en œuvre l'interface Runnable et exécuter un nouveau thread.

Nous avons vu des exemples d'exceptions non vérifiées, qui sont vérifiées au moment de l'exécution, dans les leçons précédentes. Exemple: lorsque vous tentez de diviser par 0!
<<<

```
Java
1  public class MyClass {
2      public static void main(String[ ] args) {
3          int value = 7;
4          value = value / 0;
5      }
6  }
```

/*Exception in thread "main" java.lang.ArithmeticException: / by zero at MyClass.main(MyClass.java:4)*/

Runtime vs. Checked Exceptions

Types d'exceptions

Il existe deux types d'exceptions, cochées et non cochées (également appelées runtime). La principale différence est que les exceptions cochées sont vérifiées lors de la compilation, tandis que les exceptions non cochées sont vérifiées au moment de l'exécution.
Comme mentionné dans notre leçon précédente, Thread.sleep () lance une exception InterruptedException. Ceci est un exemple d'exception vérifiée. Votre code ne compile pas tant que vous n'avez pas traité l'exception.

```
public class MyClass {
  public static void main(String[] args) {
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      //some code
    }
  }
}
```

```
1  public class MyClass {
2      public static void main(String[] args) {
3          Thread.sleep(1000);
4  }
5  }
```

Unhandled exception type InterruptedException
2 quick fixes available:
Add throws declaration
Surround with try/catch

ArrayList

L'API Java fournit des classes spéciales pour stocker et manipuler des groupes d'objets.
Une telle classe est ArrayList. Les tableaux Java standard sont d'une longueur fixe, ce qui signifie qu'après leur création, ils ne peuvent ni se dilater ni se rétrécir.
D'autre part, ArrayLists sont créés avec une taille initiale, mais lorsque cette taille est dépassée, la collection est automatiquement agrandie.
Lorsque des objets sont supprimés, ArrayList peut diminuer de taille. Notez que la classe ArrayList se trouve dans le paquetage java.util. Il est donc nécessaire de l'importer avant de l'utiliser.
Créez une ArrayList comme vous le feriez pour n'importe quel objet.

```
import java.util.ArrayList;
//...
ArrayList colors = new ArrayList();
```

Vous pouvez éventuellement spécifier une capacité et le type d'objets que la table ArrayList contiendra:

```
ArrayList<String> colors = new ArrayList<String>(10);
```

Le code ci-dessus définit un ArrayList de Strings avec 10 comme taille initiale.

ArrayLists stocke des objets. Ainsi, le type spécifié doit être un type de classe. Vous ne pouvez pas passer, par exemple, int comme le type des objets. Utilisez plutôt les types de classes spéciales qui correspondent au type de valeur souhaitée, tels que Integer pour int, Double pour double et ainsi de suite.

La classe ArrayList fournit un certain nombre de méthodes utiles pour manipuler ses objets.
La méthode add () ajoute de nouveaux objets à ArrayList. Inversement, les méthodes remove () suppriment des objets de la liste ArrayList.

```
Java
1  import java.util.ArrayList;
2
3  public class MyClass {
4      public static void main(String[ ] args) {
5          ArrayList<String> colors = new ArrayList<String>();
6          colors.add("blue");
7          colors.add("blue");
8          colors.add("green");
9          colors.add("orange");
10         colors.remove("green");
11
12         System.out.println(colors);
13     }
14 }
```

D'autres méthodes utiles sont les suivantes:
- contains () Renvoie true si la liste contient l'élément spécifié
- get (int index): retourne l'élément à la position spécifiée dans la liste
- size (): renvoie le nombre d'éléments dans la liste
- clear (): Supprime tous les éléments de la liste

Remarque: Comme pour les tableaux, l'indexation commence par 0.

LinkedLists

Vous ne pouvez pas spécifier une capacité initiale pour le base liée.

```
Java
1  import java.util.LinkedList;
2
3  public class MyClass {
4      public static void main(String[ ] args) {
5          LinkedList<String> z = new LinkedList<String>();
6          z.add("blue");
7          z.add("blue");
8          z.add("green");
9          z.add("orange");
10         z.remove("green");
11         System.out.println(z);
12     }
13 }
```

La liste liée est très similaire en syntaxe à ArrayList.
Vous pouvez facilement modifier une liste ArrayList en une liste liée en modifiant le type d'objet.

Vous pouvez utiliser la boucle for optimisée pour itérer sur ses éléments.

```
12 ➔ 99 ➔ 37 ➔ ×
```

```
Java
1  import java.util.LinkedList;
2
3  public class MyClass {
4      public static void main(String[ ] args) {
5          LinkedList<String> c = new LinkedList<String>();
6          c.add("blue");
7          c.add("blue");
8          c.add("green");
9          c.add("orange");
10         c.remove("green");

11         for(String s: c) {
12             System.out.println(s);
13         }
14     }
15 }
```

Résumé:
- Utiliser ArrayList lorsque vous avez besoin d'un accès rapide à vos données.
- Utilisez une LinkedList lorsque vous avez besoin de faire un grand nombre d'inserts et / ou de suppressions.

LinkedList contre ArrayList
La différence la plus notable entre LinkedList et ArrayList réside dans la façon dont ils stockent des objets.
Le ArrayList est mieux pour stocker et accéder aux données, car il est très similaire à un tableau normal.
Le LinkedList est meilleur pour la manipulation de données, telles que l'insertion et suppression de numériques.

En plus de stocker l'objet, le LinkedList enregistre l'adresse mémoire (ou le lien) de l'élément qui la suit. Il est appelé LinkedList parce que chaque élément contient un lien vers l'élément voisin.

HashMap

Nous avons créé un HashMap avec des chaînes comme ses clés et ses nombres entiers. Utilisez la méthode get et la clé correspondante pour accéder aux éléments HashMap.

```
// Outputs 42
Java
1  import java.util.HashMap;

2  public class MyClass {
3      public static void main(String[ ] args) {
4          HashMap<String, Integer> dict = new HashMap<String, Integer>();
5          dict.put("foo", 42);
6          dict.put("bar", 12);
7          dict.put("baz", 15);
8          System.out.println(dict.get("foo"));
9      }
10 }
```

Les tableaux et les listes stockent des éléments en tant que collections ordonnées, chaque élément étant doté d'un indice entier.
HashMap est utilisé pour stocker les collections de données en tant que paires clé et valeur. Un objet est utilisé comme clé (index) à un autre objet (la valeur).
Les méthodes put, remove et get sont utilisées pour ajouter, supprimer et accéder aux valeurs dans le HashMap.

Un HashMap ne peut contenir de clés en double. Ajout d'un nouvel élément avec une clé qui existe déjà écrase l'ancien élément.
La classe HashMap fournit des méthodes containsKey et containsValue qui déterminent la présence d'une clé ou d'une valeur spécifiée.
Si vous essayez d'obtenir une valeur qui n'est pas présente votre carte, elle retourne la valeur de null.

Null est un type spécial qui représente l'absence d'une valeur.

Sets

Un set est une collection qui ne peut pas contenir des éléments dupliqués. Il modélise une collection (un jeu, "set") d'éléments mathématiques abstraits.
L'une des implémentations del'ensemble est la classe HashSet.

```
Java
1  import java.util.HashSet;

2  public class MyClass {
3      public static void main(String[ ] args) {
4          HashSet<String> set = new HashSet<String>();
5          set.add("A");
6          set.add("A");
7          set.add("C");
8          System.out.println(set);
9      }
10 }
```

Vous pouvez utiliser la méthode size () pour obtenir le nombre d'éléments dans le HashSet.

La classe HashSet ne conserve pas automatiquement l'ordre des éléments lorsqu'ils sont ajoutés. Pour commander les éléments, utilisez un LinkedHashSet, qui maintient une liste liée des éléments du jeu dans l'ordre dans lequel ils ont été insérés.

Qu'est-ce que le hachage?
Une table de hachage stocke des informations à travers un mécanisme appelé hachage, dans lequel le contenu informationnel d'une clé est utilisé pour déterminer une valeur unique appelée code de hachage.
Donc, fondamentalement, chaque élément dans le HashSet est associé à son code de hachage unique.

Vous avez pris connaissance des différents types de collections disponibles en java, y compris les listes, les cartes et les ensembles. Le choix de celui à utiliser est spécifique aux données que vous devez stocker et manipuler.

Sorting Lists

Pour la manipulation de données dans différents types de collection, l'API Java fournit une classe Collections, qui est incluse dans le package java.util.
L'une des méthodes de classe Collections les plus populaires est sort (), qui trie les éléments de votre type de collection. Les méthodes de la classe Collections sont statiques, vous n'avez donc pas besoin d'un objet Collections pour les appeler.

```
Java
1  import java.util.ArrayList;
2  import java.util.Collections;

3  public class MyClass {
4      public static void main(String[ ] args) {
5          ArrayList<String> animals = new ArrayList<String>();
6          animals.add("tiger");
7          animals.add("cat");
8          animals.add("snake");
9          animals.add("bear");
10         animals.add("dog");

11         Collections.sort(animals);

12         System.out.println(animals);
13     }
14 }
```

/* Outputs:[cat, dog, snake, tiger]*/

Comme vous pouvez le voir, les éléments ont été triés par ordre alphabétique.

Vous pouvez appeler les méthodes sort () sur différents types de listes, comme Integers.

```
Java
1  import java.util.ArrayList;
2  import java.util.Collections;

3  public class MyClass {
4      public static void main(String[ ] args) {
5          ArrayList<Integer> nums = new ArrayList<Integer>();
6          nums.add(3);
7          nums.add(24);
8          nums.add(12);
9          nums.add(42);
10         nums.add(2);

11         Collections.sort(nums);
12         System.out.println(nums);
13     }
14 }
```

Autres méthodes utiles dans la classe Collections:
Max (Collection c): Renvoie l'élément maximal en c tel que déterminé par ordre naturel.
Min (Collection c): Renvoie l'élément minimal en c tel que déterminé par ordre naturel.
Reverse (List list): Annule la séquence dans la liste.
Shuffle (List des listes): Shuffle (c'est-à-dire, randomize) les éléments de la liste.

Fill in the blanks to add three items to the list, sort it, and print 2.

```
List<Integer> list = new ArrayList<Integer>();
list .add( 4);
list.add(2);
list .add(1);
Collections .sort (list);
System.out.println(list.get( 1 ));
```

Iterators

Un itérateur est un objet qui permet de parcourir une collection, d'obtenir ou de supprimer des éléments.
Avant d'accéder à une collection via un itérateur, vous devez en obtenir un. Chacune des classes de collection fournit une méthode Iterator () qui renvoie un itérateur au début de la collection. En utilisant cet objet itérateur, vous pouvez accéder à chaque élément de la collection, un élément à la fois.

La classe Iterator fournit les méthodes suivantes:
HasNext () Renvoie true si il y a au moins un élément de plus; Sinon, il renvoie false.
Next () Renvoie l'objet suivant en fait avancer l'itérateur.
Remove () Supprime le dernier objet renvoyé par next de la collection.

La classe Iterator doit être importée du paquet java.util.

```
//Outputs "fox"
Java
1  import java.util.Iterator;
2  import java.util.LinkedList;

3  public class MyClass {
4      public static void main(String[ ] args) {
5          LinkedList<String> animals = new LinkedList<String>();
6          animals.add("fox");
7          animals.add("cat");
8          animals.add("dog");
9          animals.add("rabbit");

10         Iterator<String> it = animals.iterator();
11         String value = it.next();
12         System.out.println(value);
13     }
14 }
```

It.next () retourne le premier élément de la liste, puis déplace l'itérateur vers l'élément suivant.
Chaque fois que vous appelez it.next (), l'itérateur passe à l'élément suivant de la liste.

Typiquement, les itérateurs sont utilisés dans des boucles. A chaque itération de la boucle, vous pouvez accéder à l'élément de liste correspondant.

```
Java
1  import java.util.Iterator;
2  import java.util.LinkedList;

3  public class MyClass {
4      public static void main(String[ ] args) {
5          LinkedList<String> animals = new LinkedList<String>();
6          animals.add("fox");
7          animals.add("cat");
8          animals.add("dog");
9          animals.add("rabbit");

10         Iterator<String> it = animals.iterator();
11         while (it.hasNext()) {
12             String value = it.next();
13             System.out.println(value);
14         }
15     }
16 }
```

/*fox
cat
dog
rabbit*/

(x), la boucle while détermine si l'itérateur a des éléments supplémentaires. Imprime la valeur de l'élément suivant et fait avancer l'itérateur vers le suivant.

Fill in the blanks to iterate and print all of the items in the list of Integers.

```
Iterator< Integer > it = list .iterator ();
while (it .hasNext, ());
  System.out.println(it .next, ());
}
```

Working with Files

Utilisation des fichiers
Le package java.io comprend une classe File qui vous permet de travailler avec des fichiers.
Pour démarrer, créez un objet File et spécifiez le chemin du fichier dans le constructeur.

```
import java.io.File;
...
File file = new File("C:\\data\\input-file.txt");
```

Avec la méthode exists (), vous pouvez déterminer si un fichier existe ou non.

```
import java.io.File;

public class MyClass {
  public static void main(String[ ] args) {
    File x = new File("C:\\sololearn\\test.txt");
    if(x.exists()) {
      System.out.println(x.getName() + "existsl");
    }
    else {
      System.out.println("The file does not exist");
    }
  }
}
```

Le code ci-dessous affiche un message indiquant si le fichier existe ou non au chemin spécifié.

La méthode getName () renvoie le nom du fichier.
Notez que nous avons utilisé des doubles barres obliques inverses dans le chemin, car une barre oblique inverse doit être échappée dans le chemin Chaîne.

Reading a File

Les fichiers sont utiles pour stocker et récupérer des données, et il existe plusieurs façons de lire des fichiers.
L'une des méthodes les plus simples consiste à utiliser la classe Scanner du paquet java.util.
Le constructeur de la classe Scanner peut prendre un objet File comme entrée.
Pour lire le contenu d'un fichier texte sur le chemin «C:\ sololearn \ \ test.txt», il faudrait créer un objet File avec le chemin correspondant et le transmettre à l'objet Scanner.

```
try {
  File x = new File("C:\\sololearn\\test.txt");
  Scanner sc = new Scanner(x);
  while(sc.hasNext()) {
    System.out.println(sc.next());
  }
  sc.close();
} catch (FileNotFoundException e) {
  System.out.println("Error");
}
```

Nous avons entouré le code avec un bloc try / catch, car il y a une chance que le fichier n'existe pas.

La classe Scanner hérite de l'itérateur, donc l'on nous pouvons utiliser la méthode next () de l'objet Scanner pour lire le contenu du fichier.

```
try {
  File x = new File("C:\\sololearn\\test.txt");
  Scanner sc = new Scanner(x);
  while(sc.hasNext()) {
    System.out.println(sc.next());
  }
  sc.close();
} catch (FileNotFoundException e) {
  System.out.println("Error");
}
```

Le contenu du fichier est sorti mot à mot, car la méthode next () renvoie chaque mot séparément.

Il est toujours recommandé de fermer un fichier lorsque vous avez terminé de l'utiliser avec File. Nous pouvons le faire en appelant la méthode close () du scanner.

Fill in the blanks to read and print the content of the file a.txt, and then close it.

```
try {
  File f = new File("a.txt");
  Scanner sc = new Scanner (f);
  while (sc .hasNext ()) {
    String s = sc .next();
    String b = sc .next ();
    System.out.println(s + " " + b);
  }
  sc .close ();
}
catch (Exception e) {
  System.out.println("Error");
}
```

Creating & Writing Files

Formatter, une autre classe utile dans le paquetage java.util, est utilisé pour créer du contenu et l'écrire dans des fichiers.

```
import java.util.Formatter;

public class MyClass {
  public static void main(String[ ] args) {
    try {
      Formatter f = new Formatter("C:\\sololearn\\test.txt");
    } catch (Exception e) {
      System.out.println("Error");
    }
  }
}
```

Crée crée un fichier vide au chemin spécifié. Si le fichier existe déjà, il l'écrasera.

Encore une fois, vous devez entourer le code avec un bloc try / catch, car l'opération peut échouer.

Une fois le fichier créé, vous pouvez lui écrire du contenu à l'aide de la méthode format () de l'objet Formatter:

```
Java
1  import java.io.File;
2  import java.io.Formatter;
3  import java.io.FileNotFoundException;
4  import java.util.Formatter;

5  public class MyClass {
6      public static void main(String[ ] args) {
7          try {
8              Formatter f = new Formatter("test.txt");
9              f.format("%s %s %s", "1", "John", "Smith \n");
10             f.format("%s %s %s", "2", "Amy", "Brown");
11             f.close();

12             File a = new File("test.txt");
13             Scanner sc = new Scanner(a);
14             while(sc.hasNext()) {
15                 System.out.println(sc.next());
16             }
17             sc.close();
18         } catch (Exception e) {
19             System.out.println("Error");
20         }
21     }
22 }
```

La méthode format () forme ses paramètres en fonction de son premier paramètre.
% S signifie une chaîne et get est remplacé par le premier paramètre après le format. Le second% s get est remplacé par le suivant, et ainsi de suite. Ainsi, le formatrk ehs ehs y désigne trois chaînes séparées par des espaces.
Remarque: \ n \ n est le symbole de nouvelle ligne dans Windows.
Le code ci-dessus crée un fichier avec le contenu suivant:

```
1 John Smith
2 Amy Brown
```

N'oubliez pas de fermer le fichier une fois que vous avez fini d'y écrire!

Tests

Fill in the blanks to declare a function that takes two integers as arguments and prints their division. Handle the possible exception.

```
void myFunc(int a, int b) {
  try {
    System.out.println(a / b);
  }
  catch (Exception e) {
    System.out.println("Error");
  }
}
```

Fill in the blanks to define a function "myFunc", which throws an exception if its parameter is less than 0.

```
void myFunc(int arg)
  throws IllegalArgumentException
{
  if (arg < 0)
  throw new IllegalArgumentException();
}
```

```
Java
1  import java.io.File;
2  import java.io.Scanner;
3  import java.io.Formatter;

4  public class MyClass {
5      public static void main(String[ ] args) {
6          try {
7              Formatter f = new Formatter("test.txt");
8              f.format("%s %s %s", "1", "John", "Smith \n");
9              f.format("%s %s %s", "2", "Amy", "Brown");
10             f.close();

11             File a = new File("test.txt");
12             Scanner sc = new Scanner(a);
13             while(sc.hasNext()) {
14                 System.out.println(sc.next());
15             }
16             sc.close();
17         } catch (Exception e) {
18             System.out.println("Error");
19         }
20     }
21 }
```

```
1 John Smith
2 Amy Brown
```

Which two options allow you to create new threads?
Select all that apply

- extend the Thread class
- handle exceptions
- name your class "Thread"
- implement Runnable

What is the output of this code?
Array.asInteger(-34) + new Array(-34).integer();
for (int i = 0; i < 5; i++)
{
  for (int j = 0;
  Console.Write(Math.Abs(i - j) + " ");
  }
  Console.WriteLine();
}
System.out.print(x);

What is the output of the following code?
int x = 0;
while(x<5)
{x++;}
System.out.print(x);

12