**Exercice 1 : lectures de programmes étranges**

1. Indiquer ce qu'affichent les programmes suivants ?

|  |  |  |
| --- | --- | --- |
| CODE | AFFICHAGE | COMMENTAIRE |
| int output = 10;  boolean result = false; if((result) && ((output += 10) == 20)) System.out.println("True " + output); else System.out.println("False " + output); | False 10 | La condition vérifie si result vaut true et si output + 10 vaut 20.  La deuxième condition est respectée tandis que la première non. |
| int counter = 0; for (int i = 1; i < 010; i++) { counter = counter + i; } System.out.println("Result : " + counter); | 28 | Calcule la somme des i de 1 à 8  010 est en base 8  010 en base 10 = 0\*(8^2) + 1\*(8^1) + 0\*(8^0) |
| public void check(boolean isOk) { if (isOk = true) System.out.println("OK"); else System.out.println("KO"); | Erreur | Cette fonction n’est pas visible dans le main sans le mot clé « static » |
| Boolean ignore = null; if (ignore == false) { System.out.println("Do not ignore!"); } | NullPointerException | On a assigné la valeur null à l’objet de type Boolean. Puis, on a essayé d’accéder au contenu de cet objet (or qui est à null) |
| static int mystery(int x, int y) { // x et y positifs if (y == 0) return 0; else if (y % 2==0) return 2 \* mystery(x, y / 2); else return x + (2 \* mystery(x, (y - 1) / 2)); } Avec dans main : System.out.println(mystery(3,4)); | 12 | Fonction qui modélise la multiplication de 2 nombres (x et y)  Fonction récursif |

1. Qu’affiche le code suivant ?

|  |  |  |
| --- | --- | --- |
| CODE | AFFICHAGE | COMMENTAIRE |
| public class Test {  String string = "i";  void f() {  try {  string += "a";  g();  } catch (Exception e) { string += "e";  }  }  void g() throws Exception { try {  string += "b";  h();  } catch (Exception e) { throw new Exception(); }finally {  string += "d";  }  string += "3";  }  void h() throws Exception { throw new Exception();  }  void display() { System.out.println(string);  }  public static void main(String[] args) {  Test test = new Test(); test.f();  test.display();  }  }  . | Iabde | TRAITER LES EXCEPTIONS   * Dans un bloc “**try**”, on place le bout de code qui pourrait générer une exception * On capture ensuite cette exception avec « **catch** (type de l’exception) » afin de ne pas arrêter le programme et afficher le message d’erreur dans catch * Le bout de code dans le bloc « **finally** » s’exécute que l’exception soit capturée ou pas (après le try si le catch ne capture pas l’exception sinon après le catch)   LANCER DES EXCEPTIONS   * « **throws** + nom de l’exception » permet de lancer une exception avec « **throw** new nomDeLException» * Il faut ensuite utiliser un try catch pour l’utilisation du bout de code dans le throws   Throws E{  try{…  }  catch(…){  throw new E  }  }  Throws E{  throw new E  } |

1. Après avoir essayé de comprendre les manipulations mémoire de cet exercice, indiquer ce que le programme affiche.

|  |  |  |
| --- | --- | --- |
| CODE | AFFICHAGE | COMMENTAIRE |
| public class Test {  public static void f(String s) { s += "rajout";  }    public static String g(String s) { s += " texte a concatener en string";  return s;  }  public static void g(StringBuffer sb) {  sb.append(" texte a concatener en sbuffer");  }  public static StringBuffer h(StringBuffer sb) {  return (sb = new StringBuffer("simple construction"));  }  public static void main(String[] args) {  String string = "un premier texte";  f(string); System.out.println(string); string = "un deuxieme texte"; g(string); System.out.println(string);  string = g(string); System.out.println(string); StringBuffer stringBuffer = new StringBuffer("construction sbuffer"); g(stringBuffer); System.out.println(stringBuffer); stringBuffer = new StringBuffer("construction sbuffer 2");  h(stringBuffer); System.out.println(stringBuffer); stringBuffer = h(stringBuffer); System.out.println(stringBuffer);  }  } | 1. un premier texte 2. un deuxième texte 3. un deuxième texte texte a concatemer en string 4. construction sbuffer texte a concatener en sbuffer 5. construction sbuffer 2 6. simple construction | //String est non modifiable (quand on concatène avec autre chose, on crée un nouveau String et on jette l’ancien dans le garbage)  //StringBuilder et StringBuffer sont modifiables  //StringBuilder est plus rapide que StringBuffer qui lui utilise des vérous (mutli-threads)   1. Ici on alloue de la mémoire pour le mot « un premier texte » (dans le main). **Un string n’est pas modifiable donc f ne fait rien.** 2. g(String) doit soit retourner un String, soit prendre en paramètre un StringBuffer. Ici, on ne respecte aucun de ces 2 cas.   **Ici on alloue de la nouvelle mémoire pour la nouvelle valeur (comme string = new… dans le main pour « un deuxième texte »)**   1. Ici, on respecte bien le typage de g et un nouveau String est alloué   **Nouvelle allocation (dans le main )**   1. On crée un nouveau StringBuffer et on le modifier après **(sans nouvelle allocation car StringBuffer modifiable)** 2. Erreur de typage, fonction avec un type de retour 3. **Nouvelle allocation pour le nouveau mot** |