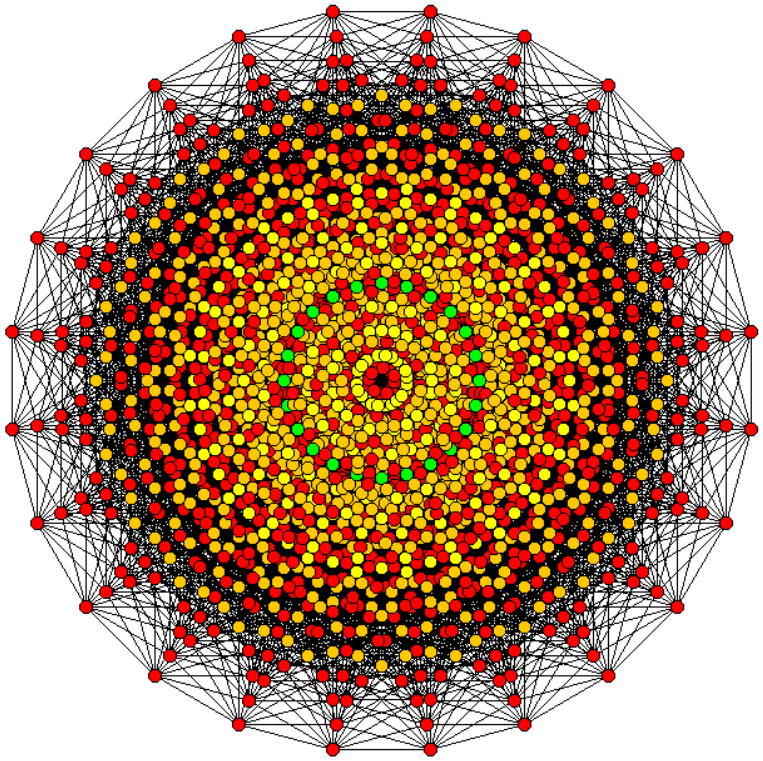


Template titre

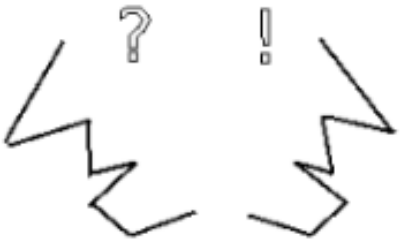
auteur

Version 1.0, 2020-04-13

Table des matieres

Présentation . . . 1
Introduction à des éléments de qualité primordiaux en génie logiciel . . . 1
  Quelques facteurs clés de qualité . . . 1
Exercice 1 Return . . . 3
  1/ Instructions *return* dans une fonction . . . 3
    Questions . . . 3
    Conclusion . . . 3
Exercice 2 Duplication . . . 4
  Duplication de code . . . 4
    Questions . . . 4
    Conclusion . . . 5
Exercice 3 Exception . . . 6
  Gestion des exceptions . . . 6
    Questions . . . 6
    Conclusion . . . 6
Exercice 4 Nommage . . . 7
  Question de nommage . . . 7
    Questions . . . 7
Exercice 5 portée de variable . . . 8
  Portée de variable . . . 8
    Questions . . . 8
Solutions . . . 9
  Éléments de réponses aux questions . . . 9

Présentation



Introduction à des éléments de qualité primordiaux en génie logiciel

Les exemples de code sont sont largement inspirés, parfois même de pur extrait, du travail en autonomie des étudiants de BTS Informatique du lycée Léonard de Vinci à Melun.

Les extraits de code présentés ci-dessous peuvent être bogués ou tout simplement « mal écrit ». Concernant ce dernier point, il est préférable de parler de code « mal conçu », qui dénote mieux l'acte de coder dans la mesure où il fusionne l'analyse et l'écriture. Le « pisseur de lignes » n'existe pas.

> Any fool can write code that a computer can understand. Good programmers write code that humans can understand
>
> — Martin Fowler

Quelques facteurs clés de qualité

Kent Beck définit dans les années 90, des règles de conception du logiciel; par ordre décroissant :

Le code :

1. passe les tests (l'objectif principal est que le logiciel fonctionne comme prévu et que des tests soient là pour garantir que cela se produise.)
2. révèle ses intentions (le code est la documentation, la logique métier omniprésente)
3. ne contient pas de duplication

Le système (code et tests pris dans leur ensemble) doit communiquer tout ce que vous avez l'intention de communiquer. Le système ne devrait contenir aucune duplication de code.

Ces deux premières contraintes constituent la règle : **Une Fois et Une Seule**.

4. Le système doit contenir un nombre minimal de classes.
5. Le système doit contenir un nombre minimal de méthodes.

En respectant ces contraintes, parfois contradictoires, le développeur produit du code clair et simple : les classes ne sont pas chargées de responsabilité et les méthodes courtes (selon différents points de vue : moins de 10 lignes ou qu'importe si le contenu est clairement lisible et ne contient pas de duplication). Exceptions connues : implémentation d'un algorithme connu, recherche d'optimisation… le tout bien commenté.

Documentations :

1. eXtreme Promamming « La référence » de Kent Beck - en français chez CampusPress
2. Refactoring de Martin Folwer
3. Refactoring to Patterns de Joshua Kerievsky

Exercice 1 Return

1/ Instructions *return* dans une fonction

Soit le fichier test.php définit par les instructions suivantes :

```php
<?php
function quiEsTu($prenom, $nom) {
  $res="inconnu";
  if ($prenom == ""){
     $res="prénom inconnu";
     return $nom;
  }
  if ($prenom=="" && $nom=="") {
     return $res;
  }
  else {
     $res=$prenom." ".$nom;
  }
  return $res;
}

function test() {
  echo  "Bonjour " . quiEsTu("", "")."\n";
  echo  "Bonjour " . quiEsTu("", "tintin")."\n";
}

f();
?>
```

Questions

1. Noter le résultat produit sur la console par la ligne de commande suivante : `php test.php`
2. Relever et commenter les instructions inutiles
3. Proposer une refonte de la fonction `quiEsTu` sans en changer le comportement final (refactoring)
4. A partir de votre solution, améliorer le comportement de la fonction `quiEsTu` (maintenance corrective/évolutive).

Conclusion

Exercice 2 Duplication

Duplication de code

Extrait de code d'une application développée par un étudiant :

```
1 public class FenetrePrincipale extends Jframe [...] {
2
3 public void actionPerformed(ActionEvent evt) {
4   Object source = evt.getSource();
5   if (source==btGenerationAndQuit) {
6      //  recupération du nom du repertoire
7      repCourant = (String) repList.getSelectedItem();
8      String nomAlbum = champsNomAlbum.getText();
9
10     // recupération du nom de la source
11     pathSource = champsSource.getText(); [...]
12
13     // création de l'album
14     fileUtil.genererAlbum(repCourant, pathSource, nomAlbum);
15
16     // lancement de l'index & fermeture de l'application
17     fileUtil.launchIndex(repCourant, nomAlbum);
18     System.exit(0);
19   }
20   if (source==mnGeneration) {
21     //  recupération du nom du repertoire
22     repCourant = (String) repList.getSelectedItem();
23     String nomAlbum = champsNomAlbum.getText();
24
25     // recupération du nom de la source
26     pathSource = champsSource.getText(); [...]
27
28     // création de l'album
29     fileUtil.genererAlbum(repCourant, pathSource, nomAlbum);
30
31     // lancement de l'index
32     fileUtil.launchIndex(repCourant, nomAlbum);
33   }
34 }
```

Questions

1. Quels types de problème de visibilité et de maintenance véhicule ce code ?
2. Proposer un refactoring (une refonte du code qui préserve le comportement initial).

Conclusion

Exercice 3 Exception

Gestion des exceptions

Voici une méthode extraite d'une application développée par un étudiant :

```
public void createImgVign(File fichierSource, File fichierDestination,
    File fichierDestinationVignette) {
  BufferedImage imageSource=null;
  try {
      //récupération de l'image source
      imageSource = ImageIO.read(fichierSource);
      //copie de l'image
      ImageIO.write(imageSource, "jpg", fichierDestination);
  } catch (IOException e) {
      System.out.println("Problème lors de la copie de l'image : "+e);
  }

  try {
      //retourne une échelle pour redimensionner l'image
      double scale = ImageUtil.getScale(sliderVignette.getValue(), imageSource);
      //transformation de l'image en vignette
      imageSource=ImageUtil.scale(imageSource, scale);
      //ecriture de la vignette
      ImageIO.write(imageSource, "jpg", fichierDestinationVignette);
  } catch (IOException e1) {
      System.out.println("Problème lors de la création de la vignette : " + e1);
  }
}
```

Questions

1. Quels sont les différents types de problèmes pouvant être générés par ce code ? (3 types différents)
2. Proposer un refactoring (une refonte qui préserve le comportement).
3. Proposer une amélioration comportementale.

Conclusion

Exercice 4 Nommage

Question de nommage

Une méthode extraite d'une application développée par un étudiant : image::code-java.png[code avec pb de nommage]

Questions

1. Dans quelle mesure les commentaires vous paraissent-ils utiles ?
2. Proposer un refactoring (une refonte qui préserve le comportement).

Exercice 5 portée de variable

Portée de variable

Voici un code java :

```java
package org.vincimelun.da;

public class Exemple {
 private int indexCandidat;
 private String lesCandidats[]= {"c1", "c2", "c3", "c4", "c5"};

 public void afficheNoms(){
  for (indexCandidat=0; indexCandidat< lesCandidats.length; indexCandidat++) {
    System.out.println(lesCandidats[indexCandidat]);
  }
 }

 public void afficheNomsBis(){
  for (indexCandidat=0; indexCandidat< lesCandidats.length; indexCandidat++) {
   afficheCandidat(lesCandidats[indexCandidat]);
  }
 }

 public void afficheCandidat(String candidat){
   System.out.println("le candidat courant est : "+ candidat);
   System.out.println("le candidat suivant est : "+
     lesCandidats[indexCandidat++]);
 }

 public static void main(String[] args) {
   Exemple test = new Exemple();
   test.afficheNoms();
   test.afficheNomsBis();
 }
}
```

Questions

1. Noter le résultat produit sur la console suite à l'exécution du programme ci-dessus.
2. Identifier et corriger le bogue.

Solutions

Éléments de réponses aux questions

Les conclusions

- Multiples sorties (return)

Un principe général consiste à formuler le fait qu'une fonction, qui par définition ne dispose que d'un point d'entrée (une fonction est appelée par son nom), ne devrait avoir qu'un seul point de sortie, qu'une seule instruction de retour (sauf exception qui ne nuirait pas à la lecture du code). À noter que le point de sortie par défaut d'une procédure est la fin de se son corps (instruction return implicite).

- Duplication de code

On factorise la logique dupliquée (la séquence d'instructions) en concevant une fonction (méthode privée)

- Exception

Une fonction qui ne peut pas réaliser sa tache ne doit pas retourner à l'appelant. voir : http://profs.vinci-melun.org/profs/okpu/cours/gestionDesExceptions/

- Commentaire

Une abondance de commentaire est signe de refactoring imminent.

- Portée de variable

Une variable utilisée pour des besoins locaux devrait être déclarée… localement (aux risques de devenir une zone tampon d'échange entre méthodes et provoquer des effets de bords indésirables). Une variable d'instance (un attribut, un champ) participe à la détermination de l'état d'un objet (instance). C'est une propriété qui est généralement accessible en lecture et ou écriture via des méthodes d'instance.