lab-sticc.univ-brest.fr/~babau/

Introduction à l'ingénierie du développement logiciel

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

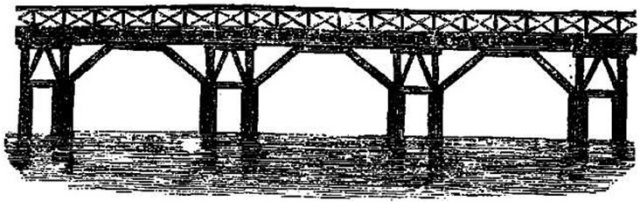
Objectifs de l'UE

- Concevoir une application
  - Analyser un cahier des charges
  - Décomposer en modules de base
  - Construire les modules de base
    - En s'appuyant sur des librairies existantes
  - Tester les modules de base
  - Intégrer pour construire une application
  - Etude de cas : placement optimisé d'antennes

Construire un pont

1

2



Pont Sublicius, Rome, 650 avant JC

3

le Ponte dei Salti (Lavertezzo, Suisse)

4

Viaduc de Millau

Mike Lehmann, Mike Switzerland March 2008

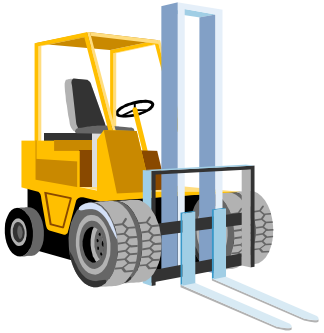
jean-philippe.babau@univ-brest.fr

Des outils pour la construction et pour l'organisation de la construction

1

2

4



3

jean-philippe.babau@univ-brest.fr

Construire une maison : documents et suivi

- Un projet : une idée de maison
- Des normes et règlements
  - Déclarations obligatoires
  - Documents de conformité
- Des plans
  - Contrat de construction de maison individuelle **« avec fourniture de plans »**
  - Schéma électrique, eau, chauffage, aération
  - Plan cadastral
- Des études techniques
  - Résistance des matériaux
  - Capacité d'isolation
- Des documents spécifiques
  - Contrats, actes, déclarations, plans, …
  - Pour le notaire, l'état, le paysagiste, les techniciens, les constructeurs, les vendeurs
- Gestion du chantier
  - Planning, revues à base des contrats et des plans

jean-philippe.babau@univ-brest.fr

Construire un logiciel

- Des systèmes de plus en plus complexe … à produire de plus en plus vite
  - des systèmes personnalisables
- Des outils de conception et modélisation de plus en plus complexes
- Des schémas hétérogènes et spécifiques
- Des acteurs divers et différents aux points de vue divers et différents

- Faire les bons choix au bon moment
- Intégrer une vision Métier **et** une vision Informatique

- Maitriser la gestion de projet informatique
  - Processus spécifiques

- Des documents pour contractualiser et communiquer
- Des outils pour le suivi
  - couts, délais, interaction avec les divers partenaires

Maitrise du développement

- En 2004, le Standish Group a évalué à 34% la part des projets qui aboutissent dans les conditions prévues, soit 18% de plus qu'il y a 10 ans.
- 15% sont arrêtés avant la fin, soit 16% de moins qu'il y a 10 ans.
- 51% sont en retard ou ont un coût supérieur au budget, soit 2% de moins qu'il y a 10 ans.

jean-philippe.babau@univ-brest.fr

Quantification des activités

- Répartition des activités
  - Analyse et conception 45%
  - Réalisation et tests unitaires : 35%
    - Codage 15 à 20% du total
  - Intégration et validation : 25%

- Dérives dans les estimations
  - Étude préalable : de 10 à 25 %
  - Conception : de 10% à 35 %
  - Réalisation : de 30% à 40%
  - Mise en œuvre : de 5% à 20%

Les points clés de la construction de logiciel

- Intégrer les besoins des utilisateurs
  - Relation client / fournisseur
  - Considérer l'ensemble des exigences de nature hétérogène
    - prix, délais
    - **fonctionnalités**
    - matériel
    - **IHM**
    - **performance**
    - sécurité
    - formation
    - déploiement
    - maintenance
  - Valider les solutions vis-à-vis des attentes du client

Les points clés de la construction de logiciel

- Choisir (ou développer) les bons outils
  - Pour le développement
  - Pour le déploiement
- Evaluer les solutions
- Maitriser le déroulement du projet
  - Développement
  - Déploiement
  - Maintenance corrective et évolutive

Limites de la compréhension humaine => séparation des préoccupations

- Analyser un problème complexe
  - Décomposition du problème en sous-problèmes
    - Décomposition objet (UML)
    - Analyse architecturale
  - Modularité et abstraction
  - « Separation of concern »
    - Dijskstra, " On the role of scientific thought" 1974
    - Reade, " Elements of Functional Programming “, 1989

Limites de la compréhension humaine => abstraction

- Lao Tseu
  - "*Trop de précision tue*"
- Nombre maximal de « token » dans un schéma
  - De 3 à 7 au maximum, mais plutôt 3
  - Limité par les capacités de compréhension du lecteur
    - Toujours inférieur aux capacités du producteur des modèles concernés
    - Le producteur connait son modèle, le lecteur le découvre
- Faire simple mais pas simpliste

Cahier des charges

- Le contexte
  - Rappeler les termes et les définitions du domaine
  - Décrire les acteurs
  - Donner l'objectif et les limites du logiciel
- Cahier des charges
  - Lever les ambiguïtés
  - Décrire précisément les fonctionnalités
  - Décrire précisément les données manipulées par le programme
    - En entrée et en sortie
    - Format des données, format de fichier, messages
  - Décrire des scénarios d'utilisation
    - Donner les résultats attendus

Cahier des charges

- *Scénarios*
  - Préciser ce que fait le logiciel en fonction des entrées
    - Sorties et état du système en fonction des entrées
    - Enchainement d'entrées
  - Considérer les cas limites

- *Contraintes*
  - Définir les cas traités et les cas non traités
  - Programmation offensive :
    - Un contrat est établi sur les données fournies
  - Programmation défensive:
    - Tous les cas sont traités

Cahier des charges

- Précis dès le début
  - On doit bien comprendre le besoin client
  - On ne s'engage pas sur un contrat pas clair
- Évolutif
  - Le client a le droit de changer d'avis
  - Une fonctionnalité n'est pas toujours réalisable dans le délai imparti
    - Equipe de développement peu importante, pas formée
    - Technologie complexe
- Méthodes itératives et agiles
  - Le client est fortement associé
  - Lien étroit avec les équipes de développement
  - Scénarisation du cahier des charges

Conception Objet

- Informatique.univ-brest.fr/java

- Une classe du point de vue conceptuel
  - Un aspect du problème
  - Un nom dans le cahier des charges
- Une classe du point de vue programmation
  - Des attributs
  - Des getters, des setters
  - Des constructeurs
  - Des opérations sur les attributs
  - Des méthodes

- Des outils pour l'édition, la génération de code, la documentation, le test, la gestion de versions

Conception Objet

- Des méthodes pour faciliter la conception, la maintenance et le test
  - Et la bonne prise en compte du besoin utilisateur

- Découper l'application en packages
  - Un package gère une préoccupation
- Classes simples
  - Nombre limité d'attributs et de méthodes
- Méthodes simples
  - Nombre limités de lignes et d'opérations
  - Limitations du nombre cyclomatique

- Un outil de qualité de code : Checkstyle
  - (http://checkstyle.sourceforge.net/)

Conception Objet

- Des classes orientées « données »
  - Pour modéliser une donnée ou un ensemble de données
  - Des attributs pivés

```
public class Etudiant {
        protected String name;
        protected int age;
        protected double moyenne;
        protected int nbNotes;
}

public class Formation {

        protected List<Etudiant> lesEtudiants;
}
```

  - Des getters, des setters
  - Des constructeurs par défaut
  - Génération via Eclipse : Source / Generate

Conception Objet

- Des classes orientées constructeurs
  - Des fabriques ou *factories*
  - Les constructeurs d'objets -> une classe *factory*
    - Le constructeur d'objet -> une méthode *CreateObject(param1, …);*

```
public class FabriqueEtudiant {

public Etudiant createEtudiant(String name) {
    Etudiant unEtudiant = new Etudiant();
    unEtudiant.setName(name);
    unEtudiant.setAge(0);
    return unEtudiant;
}

public Formation createFormation(String[] desNoms) {
    Formation uneFormation = new Formation();
    for (String unNom : desNoms) {
        uneFormation.addEtudiant(this.createEtudiant(unNom));
    }
    return uneFormation;
}
}
```

Conception Objet

- Des classes orientées traitements
  - Les méthodes complexes sont dans une ou plusieurs classes
  - Des utilitaires

```
public void addNote(Etudiant unEtudiant, double uneNote) {
        double moyenne = unEtudiant.getMoyenne();
        int nbNotes = unEtudiant.getNbNotes();
        moyenne = (moyenne*nbNotes+uneNote)/(nbNotes+1);
        unEtudiant.setNbNotes(nbNotes+1);
        unEtudiant.setMoyenne(moyenne);
}
```

Conception Objet

- Des classes orientées I/O
  - Des méthodes de lecture ou d'écriture dans un fichier
  - *Reader* : constructeur à partir d'un fichier
  - *Writer* : sauvegarde d'un objet dans un fichier

- Des classes orientées UI
  - Séparation des vues et du modèle (données et traitements)
  - Les vues ne manipulent que des objets d'interactions

- Des classes d'adaptation
  - Pou relier des classes entre elles
  - Façades, adaptateurs, …

Conception Objet

- Définition des packages
- Définition des classes
  - ATTENTION au nommage
  - Commentaire du code : JavaDoc

- Développement itératif et évolutif
  - Une classe et un programme de test à la fois

- Développement incrémental
  - Développer et tester l'IHM indépendamment de l'application
  - Commencer par les classes de données
    - Ajouter les fabriques et les traitements
  - Ajouter ensuite les classes pour les ensembles de données

Test

- Test unitaire
  - Tester chaque méthode de chaque classe avec des paramètres différents
  - Test boite noire : on en tient pas compte de l'implantation
  - Test boite blanche : on passe par tous les chemins d'exécution
  - Test :
    - Un jeu de donnée
    - Un résultat attendu
- Outils de test : JUnit

Prise en main d'une bibliothèque

- Installer la librairie
- Consulter l'API (Javadoc) et faire les tutoriaux
- Faire des programmes de tests
  - Pour connaitre l'API proposée
  - Pour évaluer l'API
- Faire des couches d'adaptation
  - Classes offrant des services dédiés pour l'application

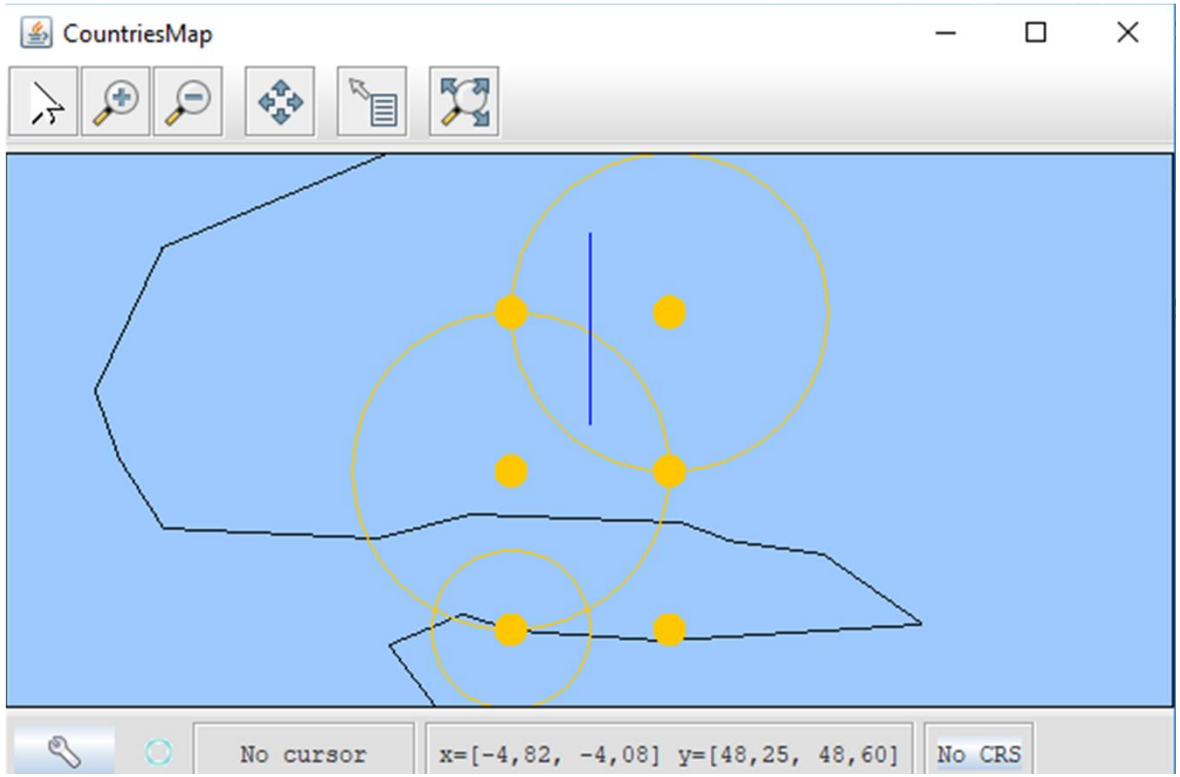
Application de placement d'antennes

- Positions à couvrir
  - Latitude [-90.0 , 90.0]
  - Longitude [-180.0,180.0]

- Antennes
  - Rayon de couverture
  - Cout

- Placement d'antennes
  - Toutes les positions sont couvertes par les antennes au moindre cout

Résultat attendu



Format des données

- Fichier d'entrée au format CSV
  - Les points : positions.csv
    - latitude (Y), longitude (X)
  - Les obstacles : obstacles.csv
    - Point départ (latitude, longitude), point d'arrivée (latitude, longitude)
  - Les antennes : antennes.csv
    - portée (rayon), cout

- Fichiers au format texte pour s'interfacer avec l'outil d'optimisation
  - Fichier d'entrée : couv.dat
    - les points <x, y>
    - les antennes <portee,cout>
    - la matrice de visibilité [0/1]
  - Fichier de sortie : couv.res
    - le cout global
    - antenne (indice), point (indice)

Cahier des charges

- Version 1 : générer la vue graphique à partir des fichiers CSV

- Version 2
  - Ajouter un menu dans l'outil pour
    - sélectionner les fichiers CSV à traiter
    - afficher le résultat
    - centrer sur le résultat
  - Vérifier le format des fichiers en entrée
  - Traiter en entrées des fichiers JSON
  - Paramétrer la couleur d'affichage des divers éléments

Cahier des charges

- Quelques questions à creuser

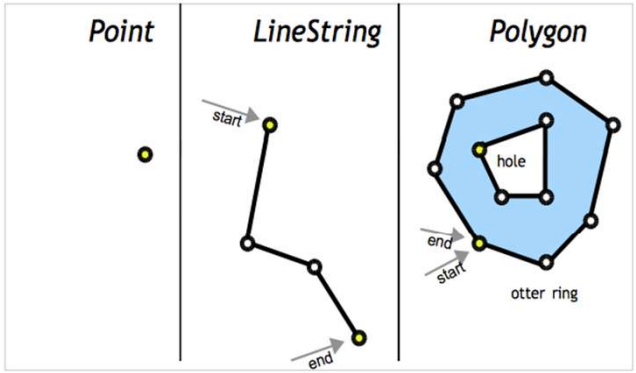
Conception

- Des packages à définir

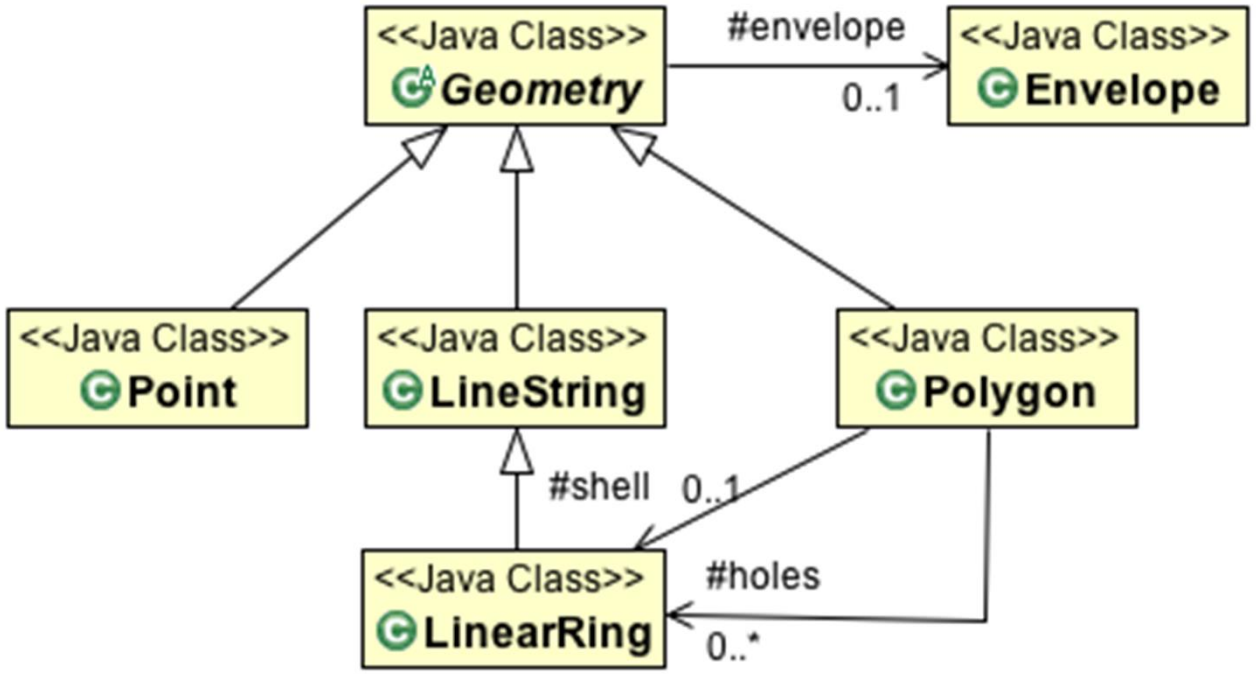
- Des classes à définir

Les librairies

- JTS pour les géométries
  - https://locationtech.github.io/jts/javadoc/org/locationtech/jts/geom/Geometry.html
- GeometryCountryMap
- Geotools
  - http://geotools.org/
  - http://docs.geotools.org/latest/userguide/tutorial/



JTS : classe *Geometry*



Construire des objets géométriques

- Utilisation une *factory*

```
GeometryFactory geometryFactory = new GeometryFactory();
Coordinate[] coordinates = new Coordinate[2];
coordinates[0] = new Coordinate(-4.5, 48.5);
coordinates[1] = new Coordinate(-4.5, 48.3);

LineString aLine = geometryFactory.createLineString(coordinates);
```

- Faire un cercle
  - faire un polygone spécifique
  - Cf. code test

- Intersections
  - Cf. javadoc de JTS

GeometryCountryMap

- Ressource nécessaire
  - Répertoire /lib
  - Répertoire *data/earth_c*

- Services

```
public class GeometryCountryMap {

    public GeometryCountryMap() {...}

    public void AddLines(List<LineString> lines, Color color) {...}

    public void AddPolygons(List<Polygon> polygons, Color color, boolean empty)
{...}

    public void Show() {...}

}
```

GeometryCountryMap

- Utilisation
  - Cf. code test

Appel exécutable en Java

- Lancement d'une commande depuis Java

```
String chemin = "C:\\workspace\\";
String[] commande = {"cmd.exe", "/C", chemin + "HelloWorld.bat"};
Process process = Runtime.getRuntime().exec( commande );
process.waitFor();
```

- Commandes
  - Cf. CplexInstall.doc

TODO List

- Préciser le cahier des charges
- Faire des programmes de tests des librairies
- Faire les classes et les programmes de tests pour la gestion des données
  - Antennes, points, obstacles
- Coder et tester la lecture et l'écriture des fichiers
  - Fichiers *.CSV*
  - Fichiers *.dat* et *.res* de l'outil
- Coder et tester les traitements
  - Calcul de la matrice de visibilité
  - Affichage des points, obstacles et placement d'antennes
  - *Appel de l'application d'optimisation*
- Intégrer et tester l'application

Documents

- J:/enseignants/jpbabau/L3
  - IDL_L3.zip : application étudiants
  - csvFiles: exemples de fichiers d'entrée du programme
  - GeotoolTests.zip : librairie GeometryCountryMap
  - cplexFiles : documentation et fichiers d'exemple de l'outil d'optimisation de placement d'antennes