Université de Bourgogne
Année universitaire 2019-2020
Info 2a — programmation objet

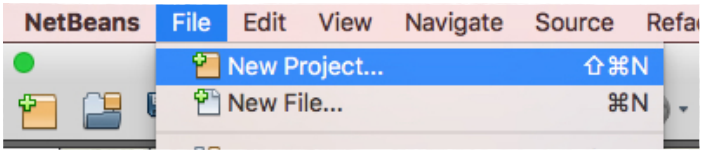
# TP 1 — *Création d'objets simples*

**Objet** : modéliser des entités simples dans le cadre objet et utiliser leur modèle informatique.

Conseils : réalisez complètement les classes et testez-les, améliorez-les… au delà des séances de TP (si vous avez un ordinateur, vous pouvez installer java et netbeans qui sont des logiciels gratuits).
Ne vous arrêtez pas dès que vous avez à peu près compris ; il n'est pas suffisant de comprendre pour pouvoir produire soi-même des programmes, il faut se les approprier. Il est aussi plus gratifiant de faire fonctionner des réalisations complètes plutôt que des programmes non finalisés aux fonctionnalités approximatives.
Si vous êtes en binôme, échangez vos rôles ; tapez à tour de rôle les programmes. Cette dernière activité favorise aussi l'appropriation.

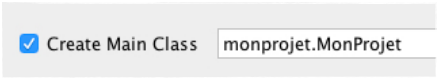
## Généralités

Pour écrire vos programmes d'info 2a sous netbeans, il faut lancer l'application, puis créer un nouveau projet :

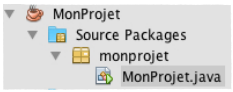



… en choisissant « application java » :

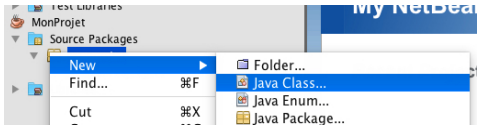

Choisissez ou créez un dossier (prenez-en un nouveau pour chaque problème de nature différente) et terminez la création du projet en n'oubliant pas de sélectionner, en bas de la fenêtre :




Le projet dispose alors d'une classe dite ici « classe principale » (ici `MonProjet`) contenant une méthode `main` (comme au premier semestre) :




Contrairement à ce qui se passait au premier semestre, l'essentiel du code de votre programme ne sera pas dans cette fonction, mais dans une ou plusieurs autres classes que vous réaliserez. La fonction `main` ne sert que d'« amorce » : elle s'exécute automatiquement lors de l'exécution du projet, mais son rôle doit se limiter à créer des instances d'autres classes, spécialisées dans certains traitements, puis à leur « donner la main ».
Pour créer ces autres classes, déroulez le menu contextuel (bouton droit) du « *package* » (ici monprojet) et choisissez java class :




Vous pouvez aussi importer des classes déjà écrites, par exemple `Lire.java` — pour gérer les entrées de données. Vous la trouverez[1] dans le dossier de partage : https://urlgo.fr/9sF3 (sous dossier code utile).

L'exécution du programme se fait en cliquant sur la flèche verte de la barre d'outils :  .
**Il ne doit y avoir qu'une seule classe par fichier** — c'est-à-dire par fenêtre de l'interface, ou encore par onglet.

---

[1] La version située dans ce dossier est un peu améliorée par rapport à celle du premier semestre. Par exemple,
`int a = Lire.i("Entrez une valeur");` est équivalent à la séquence :
`System.out.print("Entrez une valeur : "); int a = Lire.i();`

## Projets à réaliser

### *Dé*

Un dé comporte un *nombre de faces n* ($n$ peut varier d'un dé à l'autre). Il peut être utilisé pour faire un *lancer*. Par défaut, le dé comporte 6 faces.

1. Créez un projet avec sa classe principale contenant une fonction `main`, puis, ajoutez une classe `De` avec le/les attributs nécessaire/s.
2. Ajoutez la méthode `getResultat` dont l'exécution restitue une valeur aléatoire entre 1 et le nombre de faces du dé.
3. Ajoutez des constructeurs qui permettent :
   a. de créer un dé en spécifiant en paramètre le nombre de faces choisi (voir question suivante),
   b. de créer un dé sans spécifier de nombre de faces. Le nombre sera alors le nombre par défaut (6).
4. Placez dans la méthode `main` de la classe principale les instructions permettant de créer une instance de dé avec un nombre de faces choisi et de l'utiliser pour effectuer un lancer. La création du dé sera de la forme suivante :

   ```
   De de = new De(<nombre de faces>);
   ```

5. Simulez des lancements de dés avec des nombres de faces divers. Simulez aussi le lancement de plusieurs dés.

### *Caisse*

Créez un nouveau projet pour gérer une caisse (celle d'un commerçant par exemple). Une caisse comporte un *solde* et supporte des *dépôts* et des *retraits* d'argent. Réalisez une classe `Caisse` respectant ces spécifications en complétant ce qui a été vu en TD. Elle doit comporter :

1. un attribut `solde`, en euros ou en centimes d'euros.
2. les méthodes `depot` et `retrait` d'un montant. Le montant doit être positif et, en cas de retrait, il ne doit pas excéder le solde de la caisse. Sinon, aucune modification du solde ne doit avoir lieu.
3. un constructeur standard — qui crée une caisse avec un solde initial spécifié en paramètre,
4. un constructeur par défaut, correspondant au cas où le solde initial est nul.

Dans la méthode `main`, placez l'instruction suivante qui permet de créer une nouvelle caisse :

```
Caisse c = new Caisse();
```

5. Ajoutez les instructions qui permettent de tester votre caisse en effectuant des dépôts et des retraits.
6. Ajoutez l'instruction :

   ```
   c.solde = -10;
   ```

7. Exécutez le programme. Quel problème pose cette instruction ? Ajoutez des modificateurs d'accès (`private`, `public`) pour interdire d'effectuer une telle instruction et autoriser explicitement les instructions de dépôt et de retrait. Vérifiez l'effet de ces modificateurs d'accès.
8. Remplacez le contenu de la méthode `main` par un menu qui permet d'effectuer les opérations suivantes sur la caisse :
   a. Dépôt d'un certain montant,
   b. Retrait d'un certain montant,
   c. Affichage du contenu de la caisse. Quel problème pose cette fonctionnalité ?

## Si tout est fini...

### *Dés encore*

Dans le projet relatif aux lancers de dés,

1. Ajoutez à la classe `De` un attribut `dernierResultat` pour conserver la valeur du dé obtenue lors du dernier lancer (Cela permet de simuler le fait que le résultat d'un lancer de dé reste visible jusqu'au lancer suivant). Pour ce faire, ajoutez une méthode `lance` qui simule le lancer et affecte une valeur à cet attribut. La méthode `getResultat` n'aura plus alors comme rôle que de restituer le contenu de l'attribut `dernierResultat`. Écrivez les lignes suivantes dans la fonction `main` :

   ```
   De de = new De();
   System.out.println(de.getResultat());
   ```

   Quel résultat est affiché ? Pourquoi n'est-il pas satisfaisant ? Comment remédier à cela[2] ?.
2. Créez une nouvelle classe `JeuDes` pour gérer le lancement simultané de plusieurs dés identiques (même nombre de faces). Choisissez les attributs et les méthodes nécessaires. Remplacez le contenu de la fonction `main` pour créer un menu qui permet de choisir le nombre de dés et leur nombre de faces, d'effectuer un lancer de ces dés et d'afficher les résultats obtenus.

### *Caisse encore*

1. Ajoutez les éléments suivants :
   a. Une méthode `isVide` qui restitue `true` si et seulement si le solde vaut `0`,
   b. Une méthode `vide` qui vide la caisse (le solde passe à `0`).
2. Effectuez les opérations suivantes :
   a. Lancez le programme ou videz la caisse. Vérifiez que son solde est bien à 0 (`0.0`).
   b. Faites par trois fois le versement de la somme de 10 centimes d'euros (`0.1`).
   c. Effectuez le retrait de 30 centimes d'euros.
   d. Vérifiez le solde de la caisse. Que remarquez vous ?
   e. Exécutez à nouveau les opérations précédentes en faisant afficher le solde à chaque versement.
   f. Expliquez ce que vous observez. Pour mieux comprendre, allez voir là (par exemple) : http://lslwww.epfl.ch/pages/teaching/cours_lsl/intro/4.Reels.pdf.
   g. Discutez de la meilleure manière de résoudre ce problème avec votre enseignant.
3. Vous pouvez ajouter d'autres éléments à la caisse. Par exemple :
   a. La monnaie dans laquelle la caisse fonctionne — qui peut être limitée au symbole de la monnaie (€ par exemple).
   b. La liste des opérations effectuées dans l'ordre. On peut se limiter à une chaîne de caractères qui décrit la liste des opérations (intitulé de l'opération, montant éventuellement mis en œuvre et solde résultant) ou, mieux — mais beaucoup plus complexe —, créer une classe `Operation` et gérer une liste d'opérations.

## Si vous vous ennuyez vraiment...

### *Dés encore et encore*

1. *Historique des tirages.* Pour chaque dé, au lieu de conserver la valeur du dernier tirage, il est possible de conserver toutes les valeurs tirées… avec la possibilité de vider l'historique au besoin. Modifiez votre classe `De` pour assurer ces nouvelles fonctionnalités.
2. Certains dés sont « pipés », c'est-à-dire que toutes les faces n'ont pas la même probabilité de sortir. Cette différence entre les faces peut être caractérisée par un « poids » (coefficient) attaché à chaque face, qui représente sa potentialité de sortir par rapport aux autres faces. Si toutes les faces ont le même poids, le dé n'est pas pipé, si une face a un poids double de celui d'une autre face, sa probabilité de sortir est aussi double. Modifiez votre programme pour représenter ces dés particuliers. Par défaut, les dés sont normaux.

[2] Discutez avec l'enseignant responsable de votre groupe de la solution à adopter.