



1. Online shopping !

Durant ce premier exercice vous allez réaliser une partie de l'interface d'une application de vente en ligne **dont l'aspect sera aussi proche que possible de l'image ci-dessous.**

Icône de l'article courant



Il y a 2 exemplaires de cet article dans le panier

Il y a 4 articles en tout dans le panier pour un total de 148.78 €

Des boutons « << » et « >> » permettent de faire défiler les articles d'un catalogue. Un bouton ZOOM permet d'afficher une fenêtre présentant la photo de l'article courant. Un **JSpinner** permet de changer la quantité (de l'article courant) dans le panier. Enfin, un bouton vider permet de retirer tous les articles du panier.

1.1. Le modèle

 Importez le projet du TP sous eclipse.

Le noyau applicatif est composé de deux parties :

- Un catalogue, vu comme un ensemble d'articles.
- Un panier, permettant de mémoriser la quantité désirée de chaque article du catalogue.

Jetez un œil aux classes **Catalogue**, **Article** et **Panier** afin d'observer les services disponibles. **Notez** que **Catalogue** et **Panier** étendent déjà **Observable**.

1.2. L'icône et la description

Dans un premier temps, limitons nous à l'icône de l'article courant et sa description.



La méthode `main` crée déjà le catalogue et un panier initialement vide.

- ✎ **Complétez** le constructeur d'`IHMTP5` de sorte que :
- La barre de titre indique "IHM TP5".
 - La croix rouge dans la barre de titre ferme l'application.
 - L'interface soit pour l'heure constituée à gauche d'un `JLabel` présentant l'icône de l'article courant (ne pas confondre `getIcon` et `getPhoto`) et à droite d'un `JLabel` présentant la description de l'article courant.

1.3. Le ZOOM

L'appui sur le bouton ZOOM doit ouvrir une nouvelle fenêtre affichant la photo de l'article courant.



- ✎ **Créez** une classe `Zoom` correspondant à une fenêtre ne contenant que la photo (bien plus grande que l'icône) de l'article courant. Cette fenêtre aura pour titre `Zoom` et l'appui sur la croix rouge dans la barre de titre ne fermera pas l'application (mais libérera la fenêtre : `DISPOSE_ON_CLOSE`). La fenêtre `Zoom` ne doit pas pouvoir être redimensionnée par l'utilisateur.
- ✎ **Complétez** votre code de sorte que la partie gauche présente un bouton en dessous de l'icône de l'article et qu'un clic sur ce bouton affiche une fenêtre de `Zoom` sur l'article courant.

1.4. Les boutons << et >>

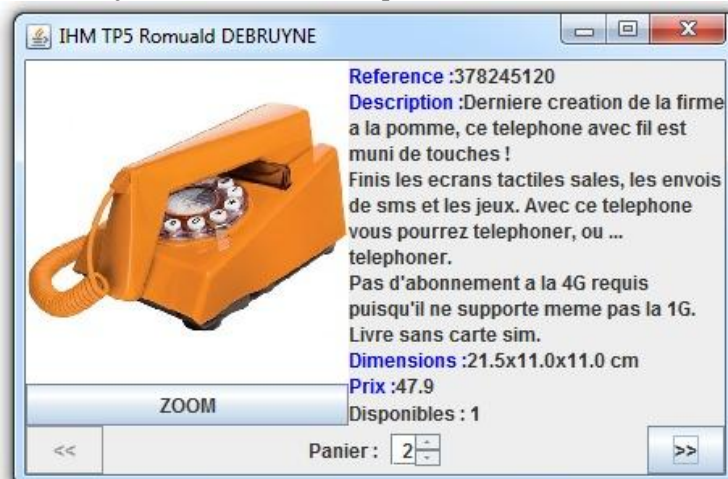
Les boutons << et >> doivent permettre de changer d'article courant, permettant ainsi la consultation de l'ensemble du catalogue.



- ✎ **Rajoutez** à votre interface deux boutons ("<<" et ">>") situés respectivement à gauche et à droite de la partie basse de la fenêtre.
- ✎ Le bouton "<<" doit permettre de passer à l'article d'index immédiatement inférieur à celui de l'article courant. La modification de l'article courant doit entraîner l'actualisation de l'icône et de la description (et le zoom doit permettre d'ouvrir une fenêtre avec la photo de l'article courant). Le bouton "<<" ne doit pas être disponible (not enabled) si l'article courant est le premier du catalogue.
- ✎ Le bouton ">>" doit permettre de passer à l'article d'index immédiatement supérieur à celui de l'article courant. La modification de l'article courant doit entraîner l'actualisation de l'icône et de la description (et le zoom doit permettre d'ouvrir une fenêtre avec la photo de l'article courant). Le bouton ">>" ne doit pas être disponible (not enabled) si l'article courant est le dernier du catalogue.

1.5. La disponibilité et le JSpinner

En dessous de la description, un **JLabel** doit indiquer le nombre d'articles de ce type disponibles et un **JSpinner** doit permettre d'ajouter/retirer des exemplaires de l'article courant au panier.



- ✎ **Ajoutez** un **JLabel** en dessous de la description. Ce **JLabel** devra indiquer la quantité disponible, c'est-à-dire la quantité en stock MOINS la quantité déjà spécifiée dans le panier. Sur l'exemple ci-dessus, il y a 3 téléphones en stock, dont 2 dans le panier, et nous ne pouvons donc en commander qu'un de plus. La disponibilité est actualisée si on change d'article courant.

✎ Ajoutez un `JLabel` et un `JSpinner` entre les boutons `<<` et `>>` comme illustré sur la page précédente. Le modèle du `JSpinner` devra faire en sorte qu'il soit possible de spécifier une quantité désirée comprise entre 0 et la quantité en stock de l'article courant. La valeur actuellement affichée via le `JSpinner` devra toujours correspondre à la quantité de l'article courant dans le panier. Le `JSpinner` est actualisé si on change d'article courant.

1.6. Le total et le bouton vider

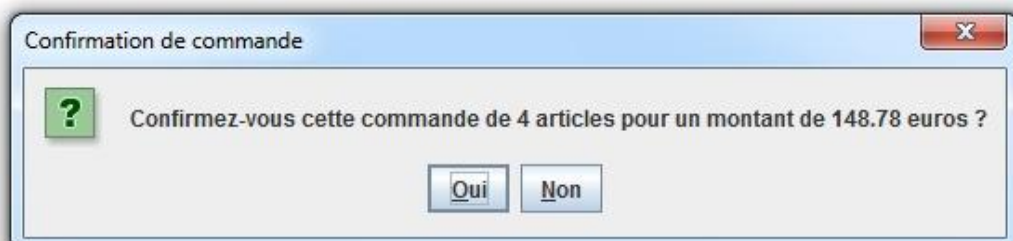
Un `JLabel` doit permettre d'indiquer le nombre total d'articles dans le panier ainsi que le montant correspondant à ces articles. De plus, un bouton `vider` doit permettre de vider le panier.



- ✎ Rajoutez à votre interface un `JLabel` et un bouton `vider` entre le `JSpinner` et le bouton `>>`.
- Le `JLabel` doit permettre d'indiquer le nombre total d'articles dans le panier ainsi que le montant correspondant à ces articles. Sur l'exemple ci-dessus, le panier comporte en tout 4 articles dont 2 téléphones. Le montant correspondant à l'ensemble de ces 4 articles est de 148.78 €.
 - Le bouton `vider` doit retirer tous les articles du panier (pas seulement ceux du type affiché → le panier ne comporte alors plus aucun article).
 - Le bouton `vider` ne doit pas être disponible si le panier est vide. On ne doit pouvoir cliquer dessus qu'à partir du moment où le panier comporte au moins 1 article.

1.7. Confirmer la commande

- ✎ Mettez en commentaire votre instruction `setDefaultCloseOperation`.
- ✎ Ajoutez un `WindowListener` à votre fenêtre qui en cas de fermeture de la fenêtre, et si le panier n'est pas vide, affichera une boîte de dialogue demandant à l'utilisateur si il confirme ou non sa commande. Si la commande est confirmée, le programme affiche « merci pour votre commande » sur la console.

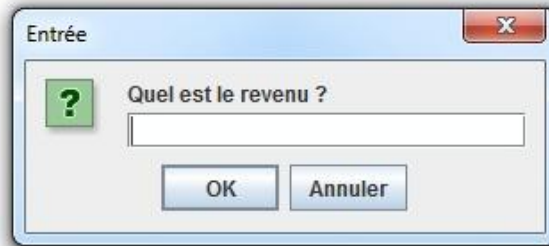


2. Prêt

Durant cette seconde partie vous allez réaliser un outil de simulation de prêt.

2.1. Saisie du revenu

L'abstraction est déjà réalisée. La classe `PlanAmortissement` permet de calculer le plan d'amortissement (le détail des échéances) correspondant à un prêt. Mais pour créer une instance de cette classe il faut indiquer un revenu. C'est en effet à partir du revenu qu'est calculé la mensualité maximale puis le montant empruntable. Avant de réaliser l'interface proprement dite, vous allez devoir faire saisir un revenu.



- Importez le projet `IHM_FIL_TP5_2` sous eclipse.
Complétez la méthode `main` de `Plan` de sorte qu'elle affiche une boîte de dialogue invitant à saisir un revenu et qu'elle répète l'affichage d'une telle boîte de dialogue tant que le montant saisi est inférieur ou égal à 0.

2.2. L'interface

Voici l'aspect qu'aura l'interface finale.

mois	restant du	interets	principal	mensualite
1	10000.0	37.66	13.12	50.78
2	9986.87	37.61	13.17	50.78
3	9973.7	37.56	13.21	50.78
4	9960.48	37.51	13.26	50.78
5	9947.21	37.46	13.31	50.78
6	9933.9	37.41	13.36	50.78
7	9920.53	37.36	13.42	50.78
8	9907.11	37.31	13.47	50.78
9	9893.63	37.26	13.52	50.78
10	9880.11	37.21	13.57	50.78
11	9866.54	37.16	13.62	50.78
12	9852.92	37.11	13.67	50.78
13	9839.24	37.06	13.72	50.78
14	9825.52	37.0	13.77	50.78
15	9811.74	36.95	13.82	50.78
16	9797.91	36.9	13.88	50.78
17	9784.03	36.85	13.93	50.78
18	9770.09	36.8	13.98	50.78

2.2.1. Le modèle

Jetez un œil à la classe `PlanAmortissement` afin d'observer les services disponibles. Notez que `PlanAmortissement` étend déjà `Observable` et notifie les éventuels observateurs en cas de modification.


2.2.2. Le plan d'amortissement

Le plan d'amortissement à proprement parlé (méthode `getPlan()`) occupe la partie centrale de l'interface. Cependant, celui-ci pouvant être particulièrement grand, la zone d'affichage doit permettre de faire défiler le contenu du plan.

Notez qu'une modification du revenu, du capital emprunté ou encore du nombre de mensualités entraîne une modification du plan d'amortissement et votre affichage du plan d'amortissement devra donc s'actualiser à chaque modification de l'abstraction.

✎ **Modifiez Plan** de sorte qu'un clic sur la croix rouge à droite de la barre de titre ferme l'application.

✎ **Modifiez Plan** en ajoutant au centre un affichage du plan d'amortissement pouvant être défilé. Veillez à ce que toute modification de l'abstraction entraîne une actualisation de l'affichage du plan.



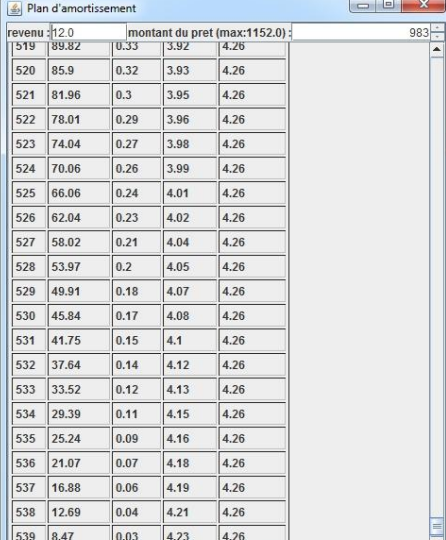
mois	restant du	interets	principal	mensualite
1	10000.0	37.66	13.12	50.78
2	9986.87	37.61	13.17	50.78
3	9973.7	37.56	13.21	50.78
4	9960.48	37.51	13.26	50.78
5	9947.21	37.46	13.31	50.78
6	9933.9	37.41	13.36	50.78
7	9920.53	37.36	13.42	50.78
8	9907.11	37.31	13.47	50.78
9	9893.63	37.26	13.52	50.78
10	9880.11	37.21	13.57	50.78
11	9866.54	37.16	13.62	50.78
12	9852.92	37.11	13.67	50.78
13	9839.24	37.06	13.72	50.78
14	9825.52	37.0	13.77	50.78
15	9811.74	36.95	13.82	50.78
16	9797.91	36.9	13.88	50.78
17	9784.03	36.85	13.93	50.78
18	9770.09	36.8	13.98	50.78
19	9756.11	36.74	14.03	50.78
20	9742.07	36.69	14.09	50.78

2.2.3. Le revenu, le capital empruntable et le capital emprunté

Au dessus du plan figurent un `JTextField` permettant de mettre à jour le revenu et un `JSpinner` afin de modifier le montant du capital emprunté. Entre les deux, un `JLabel` mentionne entre autres le montant empruntable maximal. Le pas du `JSpinner` est de 1000 euros ce qui signifie qu'un clique sur les boutons du `JSpinner` fait augmenter/diminuer le montant emprunté de 1000.

Si on tape un revenu trop faible pour pouvoir emprunter le capital emprunté courant, l'abstraction modifie le capital (et votre interface devra mentionner dans le `JSpinner` la nouvelle valeur du capital emprunté). Si dans le `JSpinner` on indique un montant supérieur au capital maximal empruntable, la valeur du `JSpinner` devient égale au capital maximal empruntable.


✎ **Modifiez Plan** de sorte à ajouter en haut de l'interface les éléments graphiques mentionnés ci-dessus. Le `JTextField` doit pouvoir modifier le revenu de l'abstraction et le `JSpinner` doit permettre de mettre à jour le capital de l'abstraction. Le `JLabel` entre le `JTextField` et le `JSpinner` doit tout comme le `JSpinner` écouter l'abstraction afin de se mettre à jour en cas de modification de ce dernier.

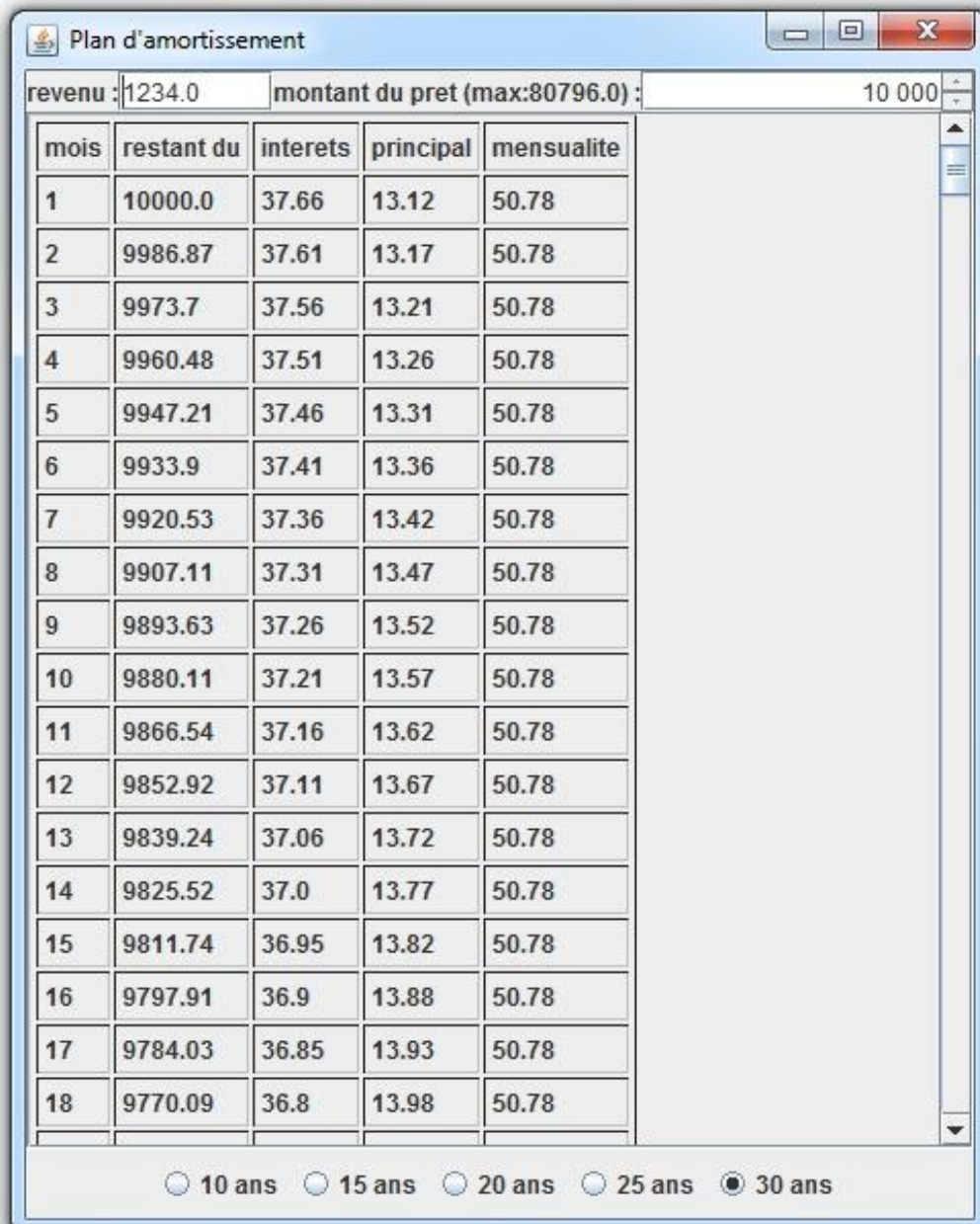


mois	restant du	interets	principal	mensualite
519	89.82	0.33	3.92	4.26
520	85.9	0.32	3.93	4.26
521	81.96	0.3	3.95	4.26
522	78.01	0.29	3.96	4.26
523	74.04	0.27	3.98	4.26
524	70.06	0.26	3.99	4.26
525	66.06	0.24	4.01	4.26
526	62.04	0.23	4.02	4.26
527	58.02	0.21	4.04	4.26
528	53.97	0.2	4.05	4.26
529	49.91	0.18	4.07	4.26
530	45.84	0.17	4.08	4.26
531	41.75	0.15	4.1	4.26
532	37.64	0.14	4.12	4.26
533	33.52	0.12	4.13	4.26
534	29.39	0.11	4.15	4.26
535	25.24	0.09	4.16	4.26
536	21.07	0.07	4.18	4.26
537	16.88	0.06	4.19	4.26
538	12.69	0.04	4.21	4.26
539	8.47	0.03	4.23	4.26

2.2.4. Le nombre de mensualités

En dessous du plan d'amortissement figurent des boutons radio. Ceux-ci permettent de sélectionner une durée de prêt de un 10, 15, 20, 25 ou 30 ans (attention, une durée de prêt s'exprime en **mois**). On ne peut bien sûr sélectionner qu'une seule durée de prêt à la fois. Ces boutons modifient le modèle mais doivent également observer le modèle pour modifier leur aspect (disponible ou non). La méthode possible de `PlanAmortissement` devrait vous être utile afin de savoir si une durée de prêt est possible (le bouton doit être disponible) ou non (le bouton doit être grisé).

 **Modifiez** `Plan` de sorte à ajouter en bas de l'interface les boutons radios évoqués ci-dessus. Ces boutons doivent modifier la durée du prêt et doivent devenir disponibles/non disponibles (grisés) en fonction de l'état du modèle.



The screenshot shows a window titled "Plan d'amortissement". At the top, there are input fields for "revenu : 1234.0" and "montant du pret (max:80796.0) : 10 000". Below these is a table with 5 columns: "mois", "restant du", "interets", "principal", and "mensualite". The table contains 18 rows of data. At the bottom of the window, there are five radio buttons for loan duration: "10 ans", "15 ans", "20 ans", "25 ans", and "30 ans". The "30 ans" button is selected.

mois	restant du	interets	principal	mensualite
1	10000.0	37.66	13.12	50.78
2	9986.87	37.61	13.17	50.78
3	9973.7	37.56	13.21	50.78
4	9960.48	37.51	13.26	50.78
5	9947.21	37.46	13.31	50.78
6	9933.9	37.41	13.36	50.78
7	9920.53	37.36	13.42	50.78
8	9907.11	37.31	13.47	50.78
9	9893.63	37.26	13.52	50.78
10	9880.11	37.21	13.57	50.78
11	9866.54	37.16	13.62	50.78
12	9852.92	37.11	13.67	50.78
13	9839.24	37.06	13.72	50.78
14	9825.52	37.0	13.77	50.78
15	9811.74	36.95	13.82	50.78
16	9797.91	36.9	13.88	50.78
17	9784.03	36.85	13.93	50.78
18	9770.09	36.8	13.98	50.78

☐ 10 ans
 ☐ 15 ans
 ☐ 20 ans
 ☐ 25 ans
 ☒ 30 ans

3. Un nouveau composant

Durant cette dernière partie vous allez réaliser un nouveau composant en spécialisant un `JPanel`.

3.1. Contexte

Nous avons réalisé une application permettant de récupérer les informations de la TAN (transports en commun nantais) afin de calculer l'itinéraire le plus court entre deux stations (le logiciel de la TAN ne retournant pas toujours l'itinéraire le plus court contrairement à ce qui est annoncé). Dans le cadre de ce TP nous ne verrons pas toute la partie abstraction mais nous nous focaliserons sur la réalisation de l'interface.

Pour l'interface de notre application une boîte de dialogue (`DialogMap`) a été réalisée. Cette dernière n'est composée que d'un seul composant (mais nous aurions pu en rajouter d'autres) : une `Map`, laquelle est une spécialisation d'un `JPanel`. C'est sur la réalisation de la classe `Map` que vous allez porter vos efforts. `Map` peut-être utilisé de deux façons :

- Soit pour afficher un itinéraire, auquel cas une liste des stations est fournie au constructeur.
- Soit pour permettre la saisie de stations (qui seront le départ, une ou des éventuelles stations intermédiaires, et l'arrivée du trajet à calculer).

Dans les deux cas l'essentiel du composant consiste à afficher le plan du réseau de la TAN. Le problème est que ce plan est très grand. Si on affiche l'intégralité du plan dans une fenêtre les noms des stations sont illisibles. Il nous faut donc prévoir la possibilité de zoomer/dézoomer et de déplacer le plan.

3.2. C'est à vous de jouer



Importez le projet `IHM_FIL_TP5_3` sous eclipse.

Le plan est affiché en `(this.x, this.y)`. Nous voulons toujours exploiter au maximum l'espace affichable du `JPanel` et l'intégralité de la `Map` sera recouverte par la carte. La fenêtre et le plan de la Tan n'ayant souvent pas le même ratio, la carte une fois dézoomée au maximum aura une de ses dimensions égale à celle de la fenêtre et débordera dans l'autre dimension (on aurait pu au contraire choisir l'option d'afficher des « bords blancs », mais ce choix n'a pas été retenu). La carte de la Tan dépasse donc généralement du cadre du `JPanel` et en dragant la souris nous pourrions déplacer la carte dans le composant. Les coordonnées `(this.x, this.y)` devront être bornées afin de ne jamais voir autre chose que le plan de la Tan dans la `Map`.

La molette de la souris est utilisée pour zoomer/dézoomer. Le niveau de zoom maximum consiste à afficher la carte dans sa vraie taille (1 pixel de l'image = 1 pixel à l'écran), tandis que le zoom minimal consiste à avoir une des dimensions de la carte qui coïncide avec une des dimensions de la `Map` (en fonction des ratios).



Suivez les indications laissées en commentaires en haut de la classe `Map`.