

CODENAME ONE

Développer en Java

pour iOS

Android

BlackBerry

et Windows Phone

Prefacé par
Shai Almog et Chen Fishbein
co-fondateurs de Codename One

Eric Dodji Gbofou

Codename One

Développer en Java pour iOS, Android,
BlackBerry et Windows Phone

par

Eric Dodji Gbofу



Codename One - Développer en Java pour iOS, Android, BlackBerry et Windows Phone
par Eric Dodji Gbofу

ISBN (EPUB) : 978-2-8227-0346-8

Copyright © 2015 Éditions D-BookeR

Tous droits réservés

Conformément au Code de la propriété intellectuelle, seules les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective ainsi que les analyses et les courtes citations dans un but d'exemple et d'illustration sont autorisées. Tout autre représentation ou reproduction, qu'elle soit intégrale ou partielle, requiert expressément le consentement de l'éditeur (art L 122-4, L 122-5 2 et 3a).

Publié par les Éditions D-BookeR, 5 rue Delouvain, 75019 Paris

www.d-booker.fr

contact@d-booker.fr

Les exemples (téléchargeables ou non), sauf indication contraire, sont propriété des auteurs.

Conception de la couverture : d'après une création de Marie Van Der Marlière (www.marie-graphiste.com)

Mise en page : générée sous Calenco avec des XSLT développées par la société NeoDoc (www.neodoc.biz)

Édition : 1

Version : 1.0

À propos de l'auteur

Eric Dodji Gbofu

Développeur d'applications depuis onze ans et amoureux du langage C++, je fais toutes mes applications avec ce langage. Il y a quelques années, je me suis intéressé à la programmation mobile et à cette époque, le langage qui régnait en maître dans ce domaine était Java. La plupart des téléphones un peu élaborés étaient sous J2ME et pour y développer des applications, je devais me mettre à Java et c'est ce que j'avais fait.

L'une des limites du développement J2ME à l'époque était son API graphique qui était vraiment pauvre en composants. C'est ainsi qu'en 2008, je tombais par un pur hasard sur la bibliothèque LWUIT qui sortait la même année chez la société Sun Microsystem. LWUIT permettait de créer des interfaces graphiques complètes, personnalisables, visuellement avancées et avec des animations. J'ai alors commencé à l'utiliser, et je l'appréciais vraiment. Étant déjà un utilisateur de LWUIT (ce dernier ayant servi plus tard à créer les bases de Codename One), la migration vers Codename One s'est faite ensuite naturellement pour moi.

Je maintiens un [site en français sur Codename One](#), où vous trouverez des articles, interviews et cours vidéo. Il est possible de me contacter via ce site pour participer à son contenu et à la construction d'une vraie communauté francophone autour de ce framework unique en son genre.

Aujourd'hui, je dirige une startup de développement de logiciels ([Yatasoft](#)) que j'ai cofondée et toutes nos applications mobiles sont conçues avec Codename One. J'enseigne aussi la programmation à des étudiants depuis huit ans et j'espère avec ce livre leur faire découvrir les plaisirs de la programmation mobile.

Bonne lecture et bon code ;-)

Préface

par Shai Almog et Chen Fishbein

Co-fondateurs de Codename One

Retour en 2005. Quand les bases de Codename One ont été posées, l’iPhone (annoncé en 2007) n’existait pas. C’était aussi le cas d’Android (dont la bêta est sortie plus tard dans la même année) et le marché du mobile était très différent. Toutefois, les problématiques liées à la diversité des appareils, à la portabilité de Java et à l’effet moteur de la portabilité se posaient déjà dans les mêmes termes.

Quand l’iPhone est sorti, nous l’avons tout de suite perçu comme une évolution majeure de l’industrie du mobile. Nous nous sommes d’emblée alignés sur ses concepts de base tels que le remplacement des barres de défilement par une simulation de “feuilletage” et les interfaces tactiles. Nos supérieurs à Sun et à Oracle en comprenaient la valeur, mais la politique, la bureaucratie et l’incompréhension du marché général nous ont empêchés de vraiment exploiter le potentiel de ce que nous avions créé. Finalement, nous avons compris que la seule façon de réaliser notre vision était de démissionner et de créer notre propre entreprise.

Cette démission fut libératrice après toutes ces années de bureaucratie et de restrictions. Nous avions soudainement la possibilité d’imaginer le produit dans son intégralité. Il a fallu un certain temps cependant pour y arriver parce que notre vision initiale était brouillée par ce que nous avions vécu. Notre première mouture était si complexe, si saturée de détails, qu’elle en devenait inutilisable. Puis, un soir, alors que nous rentrions du travail, nous avons eu une conversation tard dans la nuit, et les pièces se sont soudainement mises en place. Les serveurs de compilation seraient les pièces maîtresses de l’architecture de Codename One. Jusque-là, nous les considérions comme une simple fonctionnalité tout en leur accordant un rôle important. En modifiant notre approche et en simplifiant le produit, nous avons réussi à créer une expérience de développement unique qui garantit à la fois le contrôle et la flexibilité. Nous avons finalement compris que la devise de Java *Write Once, Run Anywhere* [écrire une fois, exécuter partout] devait s’adapter à l’évolution du temps et que c’était le seul moyen raisonnable d’atteindre notre objectif.

Développer une entreprise basée sur le cloud comme Codename One a été un défi tant du point de vue technologique que celui de la création d’une communauté. La plupart de notre temps est consacré à apporter de l’aide et à construire la communauté des développeurs qui constitue véritablement l’épine dorsale de Codename One. L’évolution du projet peut être entièrement attribuée à cette dernière. Nous voyons apparaître des communautés de développeurs dans le monde entier, à des endroits où nous n’avons jamais été et dont nous ne parlons pas la langue. C’est passionnant de voir ainsi un produit si complexe être adopté avec une telle envergure mondiale. Malheureusement, à cause de la barrière de la langue, c’est assez difficile d’atteindre tout le monde…

Voilà l’une des raisons pour lesquelles nous sommes si enthousiastes à propos de ce livre. La communauté française de Codename One a été active dès le début avec certains

développeurs notables dans divers pays. Jusqu'à présent, nous ne pouvions les renvoyer qu'à des ressources en anglais. Nous espérons que ce livre va permettre à Codename One de s'ouvrir à d'autres développeurs, qui ont peut-être eu par le passé des difficultés à le prendre en main et pourront désormais aborder le framework de manière plus naturelle.

Pour démarrer avec Codename One, vous devez garder trois choses en tête :

1. Il existe de nombreux exemples et démos à parcourir. C'est l'une des façons les plus simples pour débuter avec une nouvelle plateforme.
2. Codename One a ses particularités et vous aurez besoin de repenser beaucoup de choses. Cela peut sembler étrange au début, mais vous comprendrez que tout se ramène à la taille du code et à la compatibilité entre divers systèmes mobiles ; chose qui est évidemment assez difficile. Vous devez donc être patient.
3. Vous n'êtes pas seul ! La communauté des développeurs de Codename One est remarquablement active et nous essayons d'aider tout le monde. Le forum principal (en anglais) est très actif, mais il y a aussi une communauté française naissante qui est désireuse d'aider.

Nous espérons que vous trouverez dans Codename One comme dans ce livre des outils précieux dans vos efforts de développement mobile.

Tel-Aviv, mai 2015

Je dédie ce livre à moi-même, à Yatasoft et à tous les développeurs passionnés qui prennent plaisir à écrire chaque ligne de code dans le but de rendre meilleure l'existence des autres.

Avant-propos

Codename One est un framework extrêmement riche. Pour s'y former, il existe aujourd'hui de nombreux articles et tutoriels vidéo, ainsi qu'un forum très actif, mais ils sont presque tous en anglais. En outre, la documentation officielle, quoique très fournie, ne couvre pas toujours (bien) tous les aspects. Il manquait donc clairement un livre.

C'est un privilège pour moi d'avoir écrit le premier livre sur ce formidable framework de développement multiplateforme. Au-delà de son apprentissage, j'espère aussi vous transmettre mon enthousiasme pour cet outil, et contribuer à mieux le faire connaître auprès des développeurs Java francophones.

1. Public visé et prérequis

Ce livre s'adresse aux ingénieurs en informatique et aux développeurs d'applications (étudiants ou professionnels) sachant utiliser le langage Java et voulant apprendre la programmation mobile à travers un framework de développement multiplateforme.

Ce livre suppose que le lecteur :

- a une connaissance (même moyenne) du langage Java et en conception d'interface graphique. Ce premier point est important parce qu'aucune initiation au langage Java ne sera faite dans le livre ;
- dispose d'une connexion internet (important pour l'installation de Codename One et pour la compilation des applications) ;
- a et sait utiliser l'un des trois environnements de développement Java suivants : NetBeans, Eclipse, IntelliJ IDEA.

Note > *NetBeans est l'environnement de développement qui sera utilisé tout au long de l'ouvrage.*

En plus du langage Java utilisé pour les exemples du livre, nous utiliserons aussi le langage web PHP pour certains exemples du chapitre [Réseau, Internet et services web](#). Pour pouvoir les tester, vous aurez besoin d'un serveur web et d'une base de données sur votre ordinateur. La manière la plus simple d'avoir ces deux éléments est de télécharger et d'installer l'un des logiciels suivants qui intègrent ces deux applications (serveur web Apache et base de données MySQL) : EasyPHP, Wamp, Xampp. Si vous n'êtes pas un adepte de PHP, vous pourrez adapter les exemples à un autre langage web.

2. Réglage de la largeur de l'écran

Vous trouverez de nombreux exemples de code, formatés dans une police à chasse fixe. Afin d'éviter des retours à la ligne inopportun à l'intérieur d'une ligne de code, la longueur maximale des lignes de code a été fixée à 65 caractères, une valeur suffisamment basse pour être affichée sur la plupart des supports, tout en étant suffisante pour que le code puisse être correctement formaté.

Toutefois, il est possible que sur votre support la largeur maximale affichable soit inférieure à la limite fixée. Le paragraphe test ci-dessous permet de vérifier votre affichage. Il doit tenir sur deux lignes exactement :

```
000000000011111111222222223333333344444444555555556666  
0123456789012345678901234567890123456789012345678901234
```

Si ce n'est pas le cas, regardez si vous pouvez agrandir la taille de la fenêtre, diminuer la taille des marges ou diminuer la taille de la police d'affichage. Sur un téléphone portable, placez-le plutôt en mode paysage. Si vous n'y arrivez pas, ne vous inquiétez pas pour autant, la plupart des lignes de code sont inférieures à 65 caractères.

3. Sources des exemples

Les sources des exemples sont téléchargeables sur la page de [présentation du livre](#) sur le site des éditions D-BookeR, à l'onglet Compléments.

4. Accès aux vidéos

Dans la version en ligne, les vidéos sont intégrées à votre page, et votre navigateur ira chercher de lui-même le format qu'il supporte.

Dans les versions téléchargeables (EPUB, PDF), un clic sur l'image vous redirigera vers la vidéo en ligne au format MP4. Si votre navigateur par défaut ne supporte pas nativement ce format, modifiez à la main l'extension du fichier dans l'URL en remplaçant .mp4 par .webm.

Vous pouvez aussi consulter directement la [galerie en ligne des vidéos](#).

5. URL raccourcies

Dans un souci de lisibilité, et pour pouvoir les maintenir à jour, nous avons pris le parti de remplacer toutes les adresses internet par ce qu'on appelle des URL raccourcies. Une fois que vous avez accédé à la page cible, nous vous invitons à l'enregistrer avec un marque-page si vous souhaitez y revenir fréquemment. Vous disposerez alors du lien direct. Si celui-ci se périmé, n'hésitez pas à repasser par l'URL raccourcie. Si cette dernière aussi échoue, vous pouvez nous le signaler !

6. Remerciements

Un spécial grand merci à Shai Almog et Chen Fishbein pour avoir créé Codename One, à mon éditrice Patricia Moncorgé pour son accompagnement, sa confiance et son soutien, à Alfred Ketoglo et Fabrice Bouyé pour la relecture technique.

Mes autres remerciements vont à Laure Pello Sode pour la motivation qu'elle suscite en moi en plus de son sourire qui m'apaise énormément, à Judith Anthony pour sa présence, à Auguste Noamesi pour avoir déclenché mon envie d'apprendre la programmation à travers le HTML il y a onze ans de cela, à Edem Biova Gnona pour les bons moments qu'on a partagés ensemble pendant nos débuts en programmation, à Ben Lay, à Yves Yeme-Kponsou, à Yao Adodo De Souza et à Rhêma-Raphaël Agnam pour leurs encouragements permanents.

Introduction

1. Pourquoi utiliser un outil multiplateforme pour la programmation mobile ?

Depuis la sortie de l'iPhone, les smartphones sont devenus des ordinateurs à part entière. Et même si avant leur arrivée, il était déjà possible de créer des applications pour les téléphones qui existaient, faire des applications pour les smartphones a ouvert de nouvelles possibilités en termes de créativité. Aujourd'hui, il existe plusieurs systèmes d'exploitation mobiles pour ces téléphones intelligents. Même s'ils proposent tous des fonctionnalités proches, ces systèmes diffèrent totalement les uns des autres sur plusieurs points. Parmi eux, les plus populaires de nos jours sont iOS et Android. À côté d'eux, on trouve d'autres systèmes comme Windows Phone, BlackBerry OS, QNX, Firefox OS, etc.

À cause de cette diversité et des particularités de chaque système, créer des applications mobiles est devenu un vrai challenge si l'on souhaite cibler deux ou plusieurs de ces plateformes. Dans le cas d'une société qui veut créer une application pour diverses plateformes et qui a les moyens de se payer des développeurs spécialisés dans chacune d'elles, le problème ne se pose pas. Dans le cas d'une autre société, d'une petite équipe de développeurs ou encore d'un développeur indépendant qui n'a pas les moyens ni le temps, mais qui veut cibler plusieurs plateformes avec une même application, le travail risque de devenir très fastidieux. Écrire une bonne application pour une seule plateforme demande déjà beaucoup de travail. Si en plus de ça, il faut réécrire la même application pour d'autres plateformes alors on n'est pas sorti de l'auberge. La difficulté de viser différentes plateformes avec la même application réside principalement dans les quatre points suivants :

- *Le langage de programmation et l'API utilisée diffèrent totalement d'une plateforme à une autre.* En exemple, programmer pour iOS se fait en Objective-C ou en Swift, Android et BlackBerry OS se programment en Java (avec des API différentes), Windows Phone se programme en C#.
- *L'environnement de développement utilisé.* Même s'il est possible de nos jours d'utiliser un même environnement de développement pour programmer dans plusieurs langages différents, certains environnements sont souvent dédiés et plus adaptés à un langage. Ainsi, le développeur iOS utilisera de préférence l'environnement Xcode, le développeur Android utilisera Android Studio ou Eclipse, le développeur Windows Phone utilisera Visual Studio, etc.
- *L'interface utilisateur.* Chaque plateforme mobile propose une manière propre à elle de naviguer entre les interfaces, de présenter les menus, d'interagir avec une application, etc.
- *Le système d'exploitation de la machine de développement.* Aussi dommage que cela puisse être, il n'est pas possible de développer pour certains systèmes mobiles si l'on n'a pas le système d'exploitation approprié sur son ordinateur. En exemple, il faut avoir un Mac pour pouvoir créer des applications pour iOS, un PC avec Windows pour créer des applications pour BlackBerry OS et pour Windows Phone.

Depuis quelques années, des outils qualifiés d'outils de développement multiplateforme sont apparus et permettent de s'affranchir de ces quatre sources de difficultés majeures. Ces outils proposent d'utiliser un seul langage pour développer des applications fonctionnant sur plusieurs plateformes mobiles. Avec une promesse aussi alléchante, on ne peut qu'être emballé à la découverte de ces outils qui présentent malheureusement aussi leurs limites. Parmi eux, Codename One est l'un des plus aboutis, innovants et stables. Il propose d'écrire avec un code unique en Java des applications qui s'exécuteront sur cinq plateformes mobiles. Ainsi, il est possible d'affirmer qu'utiliser un outil multiplateforme permet de gagner en temps d'apprentissage, de conception et aussi en coût monétaire.

Les lignes qui suivent vont vous introduire Codename One, qui est un framework Java créé par deux ingénieurs israéliens réputés pour être des spécialistes en développement mobile bien avant même l'arrivée des smartphones. Après un [historique du framework](#), nous ferons un tour d'horizon de ses [principales caractéristiques](#) et en examinerons les [avantages](#) et les [inconvénients](#), avant d'enchaîner sur les choses sérieuses. Sur ce, bonne initiation à Codename One.

2. Historique de Codename One

Tout commence en 2007 à Sun MicroSystem (la société fondatrice de Java) avec le souci de créer une bibliothèque d'interfaces graphiques riche en J2ME. Le premier objectif de cette bibliothèque était de réduire les problèmes de fragmentation qu'il y avait au niveau des plateformes mobiles J2ME et BlackBerry. En plus de la volonté de résoudre ce problème, la bibliothèque devait aussi être flexible, riche en composants graphiques (ce qui n'était pas le cas de la bibliothèque d'interface fournie par défaut par l'API CLDC de J2ME en Java). Ainsi naquit la bibliothèque LWUIT créée par l'ingénieur israélien Chen Fishbein qui travaillait chez Sun. Étant donné que cette bibliothèque (qui est open-source) apportait une réelle solution à un problème contraignant, d'autres développeurs ont rejoint le projet. L'un d'eux était Shai Almog (développeur Java expérimenté et consultant auprès de Sun MicroSystem à l'époque). Il aidait Chen Fishbein (l'initiateur du projet) à faire évoluer LWUIT. Peu de temps après, il laissa sa casquette de consultant pour rejoindre finalement la société. Quelques années plus tard, la société Oracle racheta Sun MicroSystem, un rachat qui allait peser sur l'évolution de LWUIT qui constituera plus tard la base de développement de Codename One.

Nous sommes maintenant en 2011 et les deux ingénieurs et amis Chen Fishbein et Shai Almog ont envie de faire évoluer considérablement leur bibliothèque et de toucher aussi les plateformes mobiles présentes dans les smartphones (Android, iOS, BlackBerry, Windows phone en plus du J2ME d'origine). À cause de sa politique et pour certaines autres raisons, ils décident de quitter Oracle pour créer leur startup et réaliser leur projet. Ils présentent alors leur démission cette même année et commencent leur projet qu'ils nomment Codename One (ce qui sera aussi le nom de leur startup). Puisque LWUIT est open-source, ils clonent son code source, changent le nom des packages, font des modifications intensives, créent des portages vers d'autres systèmes mobiles et ajoutent de nombreuses fonctionnalités. Codename One est né, et sa première version bêta est lancée publiquement en janvier 2012.

Étant open-source, en plus d'avoir un système de plug-ins qui rend son évolution flexible, Codename One n'a cessé d'évoluer depuis le début de sa création. Toujours bien accueilli par les développeurs Java principalement, Codename One est actuellement un outil complet permettant aux développeurs Java de concevoir des applications mobiles multiplateformes de qualité pour les plateformes iOS, Android, Windows phone, BlackBerry et toujours J2ME (pour les développeurs ciblant les téléphones Nokia Asha pour ne citer que ça).

3. Pourquoi Codename One ?

Codename One est un framework écrit en Java permettant de faire de la programmation mobile multiplateforme. Il est open-source et se présente sous la forme d'un plug-in disponible pour les trois environnements de développement majeurs en Java (NetBeans, Eclipse, IntelliJ IDEA). Il permet de cibler cinq plateformes mobiles (iOS, Android, Windows, BlackBerry, J2ME) avec un code unique et a aussi pour particularité d'utiliser le cloud pour la compilation. Cette utilisation du cloud permet aux développeurs de s'affranchir de l'installation de divers SDK ou de posséder un système d'exploitation spécifique pour programmer des applications pour certaines plateformes mobiles. Codename One produit toujours du code natif donc il n'y a aucune raison de se soucier des problèmes de performance. Le plug-in est composé de quatre parties majeures que voici :

Une API

Cette API contient toutes les classes nécessaires à la conception d'une application mobile et est écrite en langage Java.

Un designer

Fourni sous forme de logiciel, le designer permet de concevoir visuellement une interface d'application, de gérer la traduction d'une application en diverses langues, de créer des thèmes, de manipuler des images, etc.

Un simulateur

Il permet de tester ses applications sur son ordinateur. N'étant pas un émulateur, ce simulateur est rapide à l'exécution et embarque des outils pratiques pour tester en profondeur les applications.

Un serveur de compilation dans le cloud

Ce serveur permet de compiler en ligne les applications écrites avec Codename One. Cette manière d'effectuer les compilations a ses avantages et ses inconvénients sur lesquels nous reviendrons.

L'une des grandes particularités de Codename One est son architecture dite *lightweight* qui apporte une meilleure solution aux problèmes de fragmentation des plateformes mobiles. Un composant lightweight dans ce cas-ci est un composant écrit entièrement en Java qui dessine sa propre interface tout en gérant ses propres événements et états. Cette manière de faire apporte un énorme avantage en termes de portabilité puisque le même code est exécuté sur toutes les plateformes en plus d'autres avantages. Les composants graphiques de Codename One sont infiniment personnalisables.

L'API de Codename One couvre une immense catégorie de fonctionnalités. On peut y trouver ce qu'il faut pour faire par exemple les tâches suivantes :

- l'interface graphique ;

- la manipulation de la vidéo et de l'audio (enregistrement comme affichage) ;
- le stockage ;
- l'accès à la caméra ;
- la manipulation d'une base de données SQLite ;
- la manipulation des services web ;
- le réseau ;
- l'accès au cloud ;
- la lecture des QR et Bar codes ;
- l'internationalisation et la localisation ;
- les notifications ;
- la manipulation des contacts ;
- l'accès aux pages web ;
- la monétisation ;
- l'accès aux réseaux sociaux ;
- la géolocalisation ;
- les tests unitaires ;
- la création de thèmes personnalisés ;
- et beaucoup d'autres fonctionnalités à découvrir.

Cette liste n'est pas exhaustive donc pas d'inquiétude si vous ne trouvez pas une fonction particulière non citée ci-dessus.

Codename One est orienté exclusivement vers la conception d'applications métiers et n'a pas du tout été pensé pour créer des jeux. Quelques efforts sont en train d'être faits dans ce sens, mais rien de concret n'est encore disponible de ce côté.

Pourquoi utiliser Codename One et pas les autres outils de développement mobile multiplateforme ? Qu'a-t-il de mieux que les autres outils de ce genre ? Ce qu'il faut d'abord savoir c'est que ce langage n'est pas l'outil parfait et magique sans inconvénients qui permet de tout faire en un clic. Comme pour chaque outil ou framework, celui-ci aussi a sa propre philosophie. Une fois cette dernière acquise, son utilisation devient simple et est un vrai régal.

Voici une liste de quelques avantages et inconvénients.

3.1. Avantages

- Simulateur fourni et s'adaptant automatiquement au visuel et comportement des plateformes mobiles supportées. En plus de remplir sa tâche de base, ce simulateur

contient un ensemble d'outils pratiques pour les tests avancés.

- Compilation dans le cloud. Ceci a pour avantage de se passer de l'installation et de la configuration de divers SDK sur son ordinateur. Cela évite aussi d'avoir à utiliser un système d'exploitation spécifique pour compiler pour certaines plateformes comme le fait d'avoir un Mac avant de compiler pour l'iOS ou un PC sous Windows avant de compiler pour Windows Phone ou Blackberry. Cette méthode peut aussi faciliter le travail en équipe en permettant à chaque membre d'avoir accès aux compilations des autres.
- Présence d'un éditeur visuel d'interfaces graphiques. Peu d'outils multiplateformes (et même certains outils natifs) fournissent un éditeur graphique. Cet éditeur permet de gagner un temps considérable dans la conception des interfaces d'une application, ce qui aide à se concentrer uniquement sur les fonctionnalités.
- Utilisation du langage Java, qui est un langage stable, structuré, connu et très documenté sur le web et dans les livres est un avantage non négligeable.
- Possibilité d'obtenir une interface au visuel unique sur toutes les plateformes ou une interface propre à chaque plateforme.
- Présence d'un logiciel nommé Codename One Designer. En plus de pouvoir créer une interface graphique à la souris et de créer des thèmes visuels pour une application, ce logiciel fournit d'autres possibilités permettant de simplifier diverses choses. Un chapitre entier lui est consacré dans cet ouvrage et il s'agit du chapitre [Codename One Designer](#).
- Possibilité de créer soi-même des plug-ins liés au framework et permettant d'étendre ses fonctionnalités.
- Contrairement à d'autres frameworks de développement mobile multiplateforme qui utilisent les technologies web pour l'interface graphique ou qui traduisent leur code dans le but d'utiliser les API graphiques d'origine, Codename One dessine ses propres composants graphiques quelle que soit la plateforme visée. Cela permet de résoudre les problèmes de fragmentation liés aux interfaces sur différentes plateformes.

3.2. Inconvénients

Certains des avantages de Codename One apportent aussi certains inconvénients. Les voici :

- Le fait de compiler dans le cloud est bien, mais il serait aussi intéressant et plus pratique de compiler en local sur son propre ordinateur. Même si ce n'est pas l'option fournie par défaut, il est quand même possible de le faire après quelques bidouilles qui sont qualifiées de complexes par les créateurs du framework.
- Codename One est fourni avec un simulateur et non un émulateur. N'étant pas un émulateur, les comportements des applications ne sont pas forcément reproduits

fidèlement comme sur les plateformes réelles. Cela a pour inconvénient de voir par exemple une fonctionnalité refusant de s'exécuter (ou s'exécutant mal) sur le simulateur mais s'exécutant sur la vraie plateforme et vice versa.

- Le fait que Codename One dessine ses propres composants et y applique ensuite des thèmes au lieu d'utiliser directement les composants des SDK natifs peut être vu par certains comme un inconvénient mais ce point reste quand même très discutable.
- Le fait de ne pas pouvoir effectuer de compilation si le serveur en ligne est indisponible. Ce genre de situation est rare mais peut arriver.

Comme vous pouvez le remarquer, Codename One a aussi ses inconvénients mais le fait d'être un projet open-source et d'avoir une communauté ouverte et généreuse permet d'avoir un outil très évolutif et en constante amélioration.

Démarrage

Maintenant que vous savez en quoi Codename One peut vous aider à créer la prochaine application mobile à succès, nous allons démarrer. Dans ce premier chapitre, vous apprendrez à [installer le plug-in](#) dans votre environnement de développement Java préféré. Vous écrirez une [première application de test](#) après avoir vu la [structure d'une application Codename One](#). Enfin, vous découvrirez le [processus de compilation](#) particulier des applications Codename One et le mode de fonctionnement du simulateur inclus dans le plug-in du framework.

1. Téléchargement et installation du plug-in

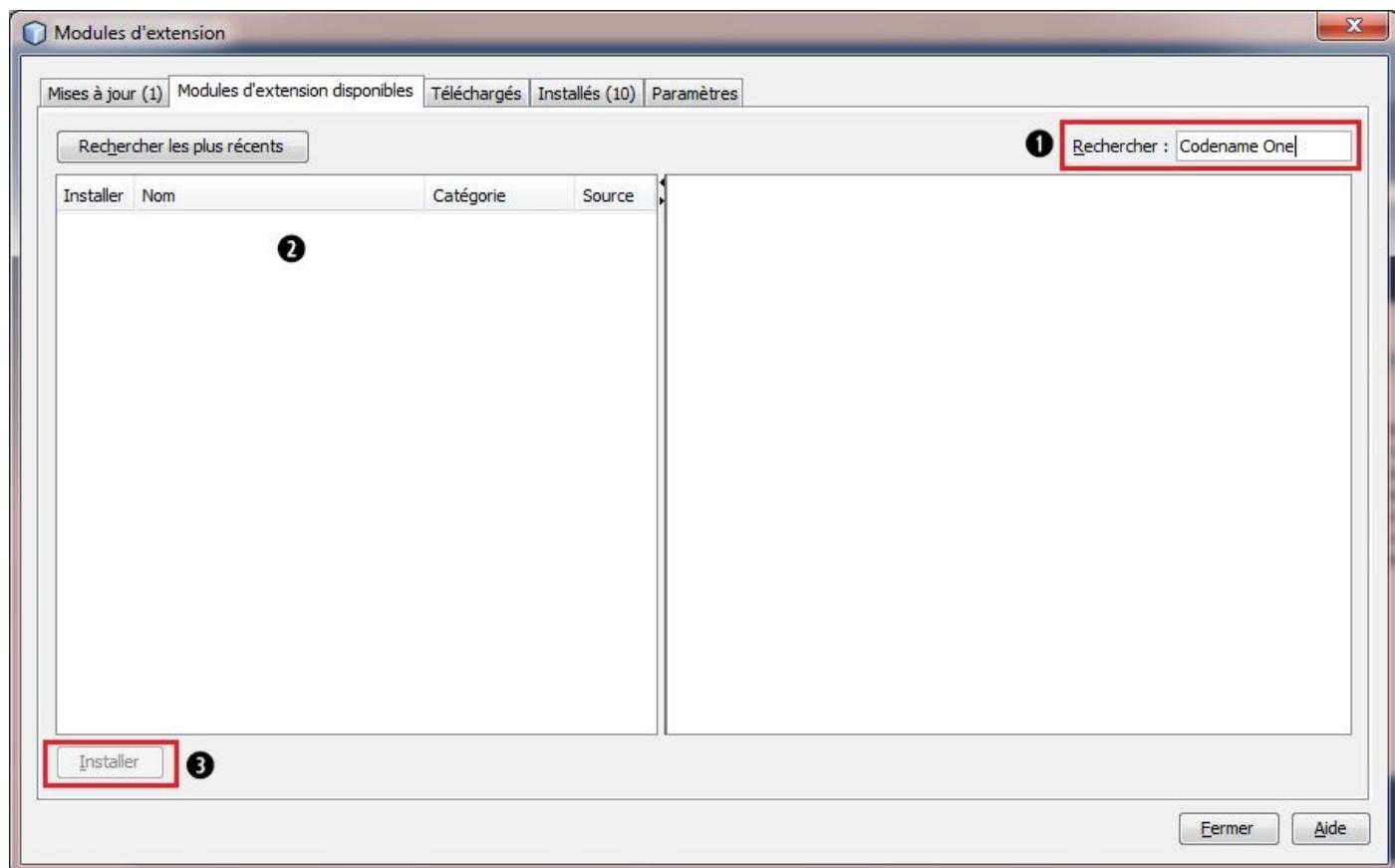
Le téléchargement du plug-in se fait directement en ligne. L'installation diffère selon les différents environnements de développement, mais reste simple que vous utilisiez NetBeans, Eclipse ou IntelliJ IDEA. Pour information, le plug-in de NetBeans a souvent la priorité sur les autres en ce qui concerne les mises à jour et c'est l'une des raisons pour lesquelles c'est l'environnement qui a été choisi pour les exemples de cet ouvrage.

1.1. Sous NetBeans

Attention > Vous devez avoir la version 7.x ou version supérieure de NetBeans pour pouvoir installer et utiliser le plug-in.

Lancez NetBeans. Allez dans le menu Outils, choisissez l'entrée Modules d'extension puis dans la zone Rechercher, saisissez Codename One (voir ❶) puis lancez la recherche. Une fois le plug-in trouvé et affiché dans la zone ❷, sélectionnez-le et cliquez sur le bouton Installer ❸ pour lancer l'installation. Suivez ensuite les instructions.

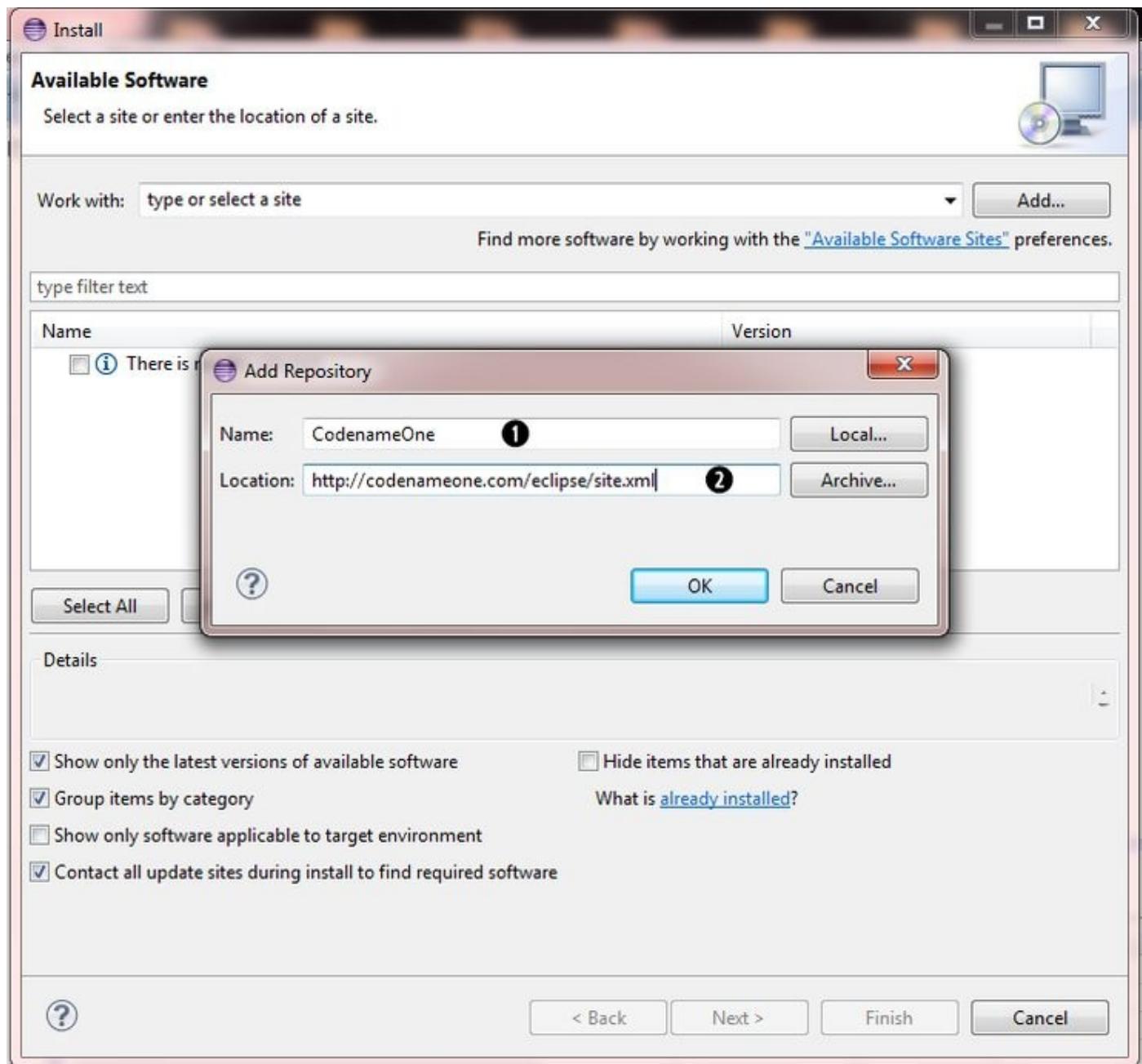
Figure 1.1 : Installation sous NetBeans



1.2. Sous Eclipse

Lancez Eclipse et cliquez sur le menu Help puis sur Install New Software (voir [Figure 1.2](#)). Cliquez sur le bouton Add. Entrez ensuite *CodenameOne* (ou ce que vous voulez) dans la zone Name **1** et dans la zone Location **2**, entrez ou copiez-collez l'url suivante : <http://codenameone.com/eclipse/site.xml>.

Figure 1.2 : Installation sous Eclipse



Sélectionnez le plug-in proposé **3** (voir [Figure 1.3](#)) et validez pour poursuivre l'installation.

Figure 1.3 : Installation sous Eclipse (suite)

Name	Version
Uncategorized	
③ CodenameOneFeature	1.0.0.201403022232

Select All Deselect All 1 item selected

1.3. Sous IntelliJ IDEA

Attention > Vous devez avoir la version 12.X ou version supérieure de IDEA pour pouvoir installer et utiliser le plug-in.

Lancez IDEA et sur l'interface de bienvenue, cliquez sur Configure puis sur Plugins. Sur l'interface qui s'affichera, cliquez sur le bouton Browser repositories pour ouvrir une nouvelle fenêtre. Dans la zone de recherche de cette fenêtre, entrez *Codename One*. Une fois le plug-in trouvé, faites un clic droit sur son nom puis choisissez Download and Install. Enfin, confirmez l'installation et le reste se fera automatiquement.

2. Structure d'une application Codename One

Comme dans chaque langage ou pour certains frameworks, un programme Codename One a aussi sa structure. Cette structure correspond au cycle d'exécution d'une application écrite avec ce framework.

```
public class NomDeLaClassePrincipale {  
    public void start() {  
        ❶ //— Code à exécuter au démarrage de l'application— } public void stop() { ❷ //—  
        Code à exécuter avant la mise en pause de l'application — } public void destroy() { ❸ //—  
        — Code à exécuter avant la fermeture de l'application— } }
```

Note > Si vous avez déjà créé des applications J2ME, vous remarquerez alors que les méthodes abstraites ci-dessus ressemblent à celles qui sont nommées `startApp()`, `pauseApp()` et `destroyApp()`.

Considérez cette classe principale comme la classe d'un projet Java classique contenant la méthode `main()`. Voici le rôle des trois méthodes de cette classe.

❶ La méthode `start()` est appelée pendant le démarrage de l'application et aussi après la reprise d'une application mise en pause. Cette méthode est le point d'entrée de votre programme. Elle peut être comparée à la méthode `main()` d'un programme Java classique.

❷ La méthode `stop()` est appelée à chaque fois qu'une application est mise en pause. Cela peut survenir lorsque l'utilisateur sort par exemple de votre application pour y revenir après, quand il reçoit un appel qui placera l'application en tâche de fond, etc. Sur certaines plateformes (iOS par exemple), l'état courant de l'application n'est pas sauvegardé par défaut quand elle est mise en pause. Cela a pour conséquence qu'elle redémarrera complètement lorsque l'utilisateur y reviendra. Pour éviter ce genre de comportement, vous pouvez sauvegarder dans une variable placée dans cette méthode l'interface courante de l'application pour pouvoir la réafficher lorsque l'utilisateur y retournera. Dans le cas d'un jeu, vous pouvez aussi utiliser cette méthode pour faire des sauvegardes et mettre le jeu en état de pause.

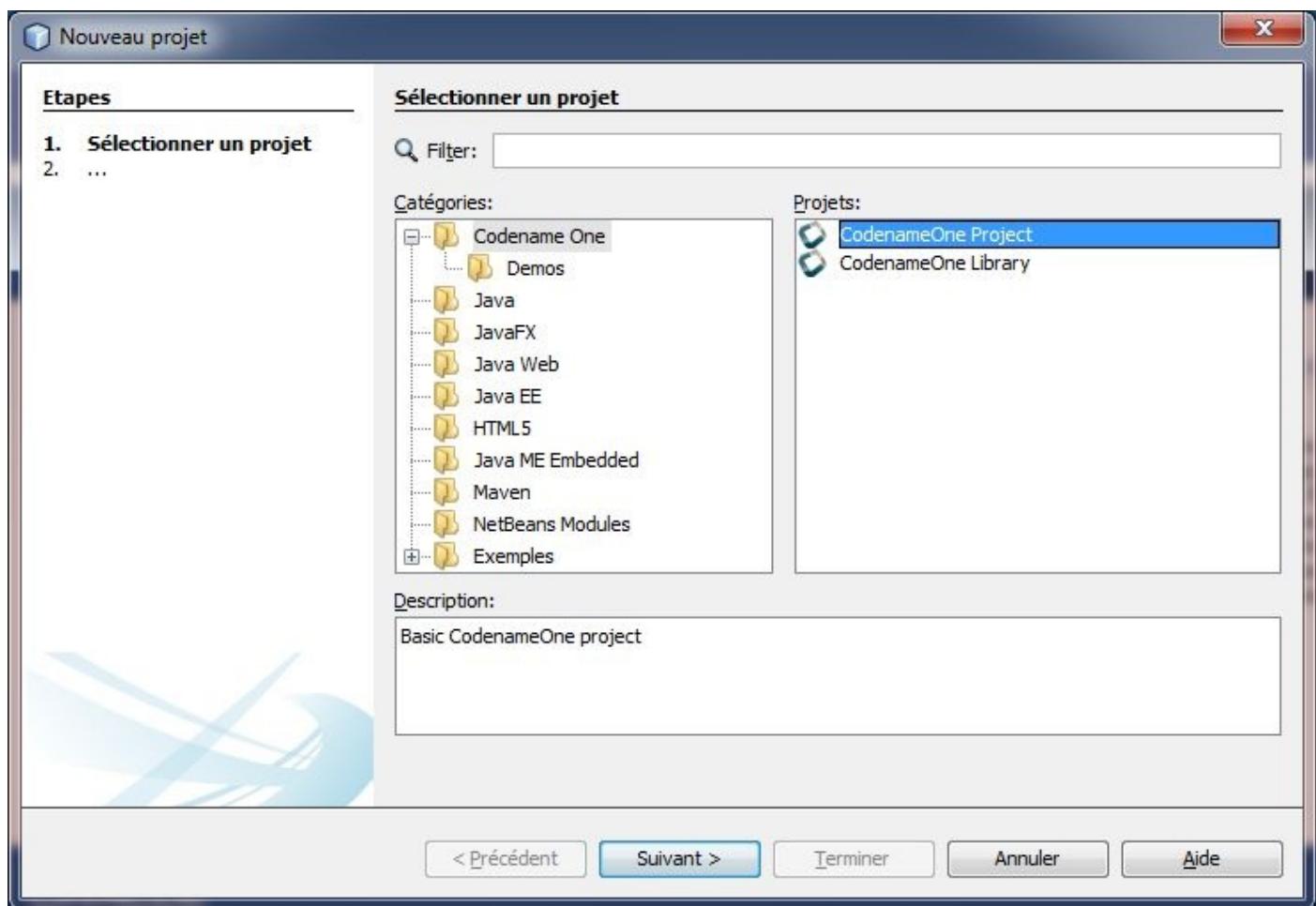
❸ La méthode `destroy()` est appelée avant la fermeture de votre application. Si vous avez des instructions à faire exécuter avant l'arrêt de l'application, mettez alors ces instructions dans cette méthode.

3. Hello Codename One

Note > À la différence des menus de votre environnement de développement, qui seront en français si vous utilisez la version française de l'IDE, ceux ajoutés par le plug-in de Codename One ne sont disponibles qu'en anglais. Dans le cas de cet ouvrage, il s'agit de NetBeans.

Il est temps de passer à la création d'une première application avec Codename One (le fameux *Hello World*) et c'est ce que nous allons faire dans cette section en créant d'abord un nouveau projet. Lancez NetBeans et cliquez sur Nouveau projet. Choisissez la catégorie Codename One puis sélectionnez CodenameOne Project.

Figure 1.4 : Crédit d'un nouveau projet (étape 1)



Cliquez sur Suivant et nommez le projet, par exemple `HelloTestApp`. Faites encore Suivant et à la dernière étape, entrez `com.codenameonefr.hellotestapp` comme nom du package et `HelloTestApp` comme nom de la classe principale qui contiendra le code de la structure de l'application. Dans la zone Theme, choisissez le thème Native Theme en cliquant dessus puis Hello World (Manual) au niveau de la zone Template.

Encadré : Explication des choix effectués

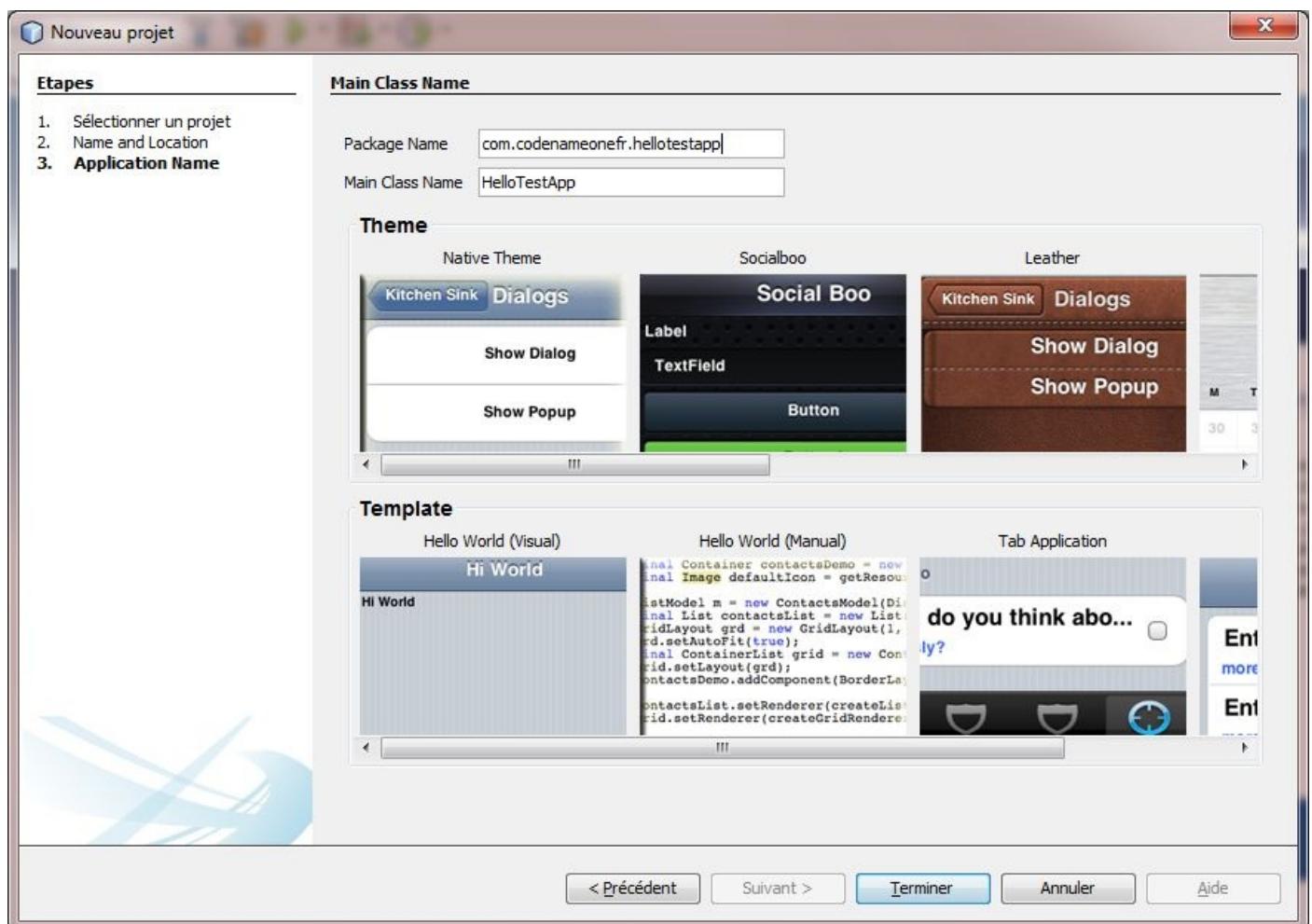
Le choix de Native Theme permet au visuel de l'application de changer et de s'adapter automatiquement aux différentes plateformes. Ainsi, au lieu d'avoir un design unique

de l'application sur les différentes plateformes, le design adoptera le rendu visuel natif propre à chaque plateforme.

Le choix du Hello World (Manual) nous permettra de créer manuellement l'interface graphique de notre application de test avec du code sans recourir à l'éditeur graphique de Codename One. C'est important parce que la structure d'un code utilisant l'éditeur graphique et celle d'un code écrit à la main diffèrent à certains niveaux. Pour utiliser l'éditeur graphique d'interface, vous devez choisir Hello World (Visual). Les autres modèles de la zone templates utilisent aussi la structure basée sur l'éditeur graphique mais nous n'allons pas commencer par ça donc gardons le choix du Hello World (Manual).

Cliquez enfin sur Terminer pour créer le projet (voir [Figure 1.5](#)).

Figure 1.5 : Création d'un nouveau projet (étape 2)



Le code généré par la création du projet sera semblable à celui-ci :

```
public class HelloTestApp {  
  
    private Form current;  
  
    public void init(Object context) {  
        try {  
            Resources theme = Resources.openLayered("/theme");  
        }  
    }  
}
```

```

❶ UIManager.getInstance().setThemeProps( theme.getTheme(
theme.getThemeResourceNames()[0]); } catch(IOException e){ e.printStackTrace(); } }
public void start() { if(current != null){ current.show(); return; } Form hi=new Form("Hi
World"); ❷ hi.addComponent(new Label("Hi World")); ❸ hi.show(); ❹ } public void
stop() { current = Display.getInstance().getCurrent(); ❺ } public void destroy() { } }

```

Avant de passer à l'explication du code généré, nous allons d'abord lancer son exécution et voir ce que ça donne dans le simulateur. Pour cela, allez sur la barre d'outils de NetBeans ou dans le menu Exécuter et cliquez sur Exécuter projet. L'exécution du code affiche le simulateur avec l'interface de l'iPhone par défaut (voir [Figure 1.6](#)).

Figure 1.6 : Exécution dans le simulateur

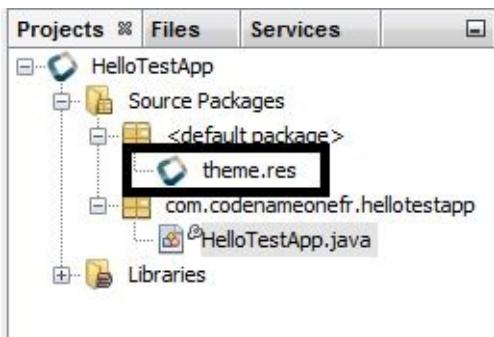


Changeons la plateforme pour voir ce que le rendu donnera sous une autre plateforme. Pour cela, allez dans le menu Skins du simulateur et cliquez sur le nom d'une autre plateforme (Nexus par exemple pour afficher le rendu des versions 2.x.x d'Android).

Décortiquons maintenant le code généré par NetBeans en commençant par la méthode `init()`. Le code à l'intérieur de cette fonction (voir ❶) permet d'abord de charger le fichier de ressources qui est un fichier d'extension `.res` (visible dans votre projet) et qui contient le thème de l'application. Ce thème est ensuite défini comme thème par défaut. Il est important de le spécifier, car plusieurs thèmes peuvent être présents dans le fichier de ressources. Les détails sur le code à l'intérieur de cette fonction seront vus au Chapitre [Codename One Designer](#). Pour l'instant, gardez-le tel quel et ne modifiez rien à cet endroit.

Note > Un fichier de ressources (voir l'encadré sur la [Figure 1.7](#)) est un fichier qui accompagne tous les projets Codename One. Il sera toujours intégré dans l'exécutable de vos applications. Il peut contenir diverses choses comme le thème de l'application, les textes de traduction dans d'autres langues, des images, des fichiers textes, des polices de caractères et bien d'autres choses que nous verrons au chapitre [Codename One Designer](#).

Figure 1.7 : Aperçu du fichier de ressources dans le projet



Pour revenir à la méthode `init()`, considérez-la comme un constructeur et faites-y des initialisations si vous voulez.

Passons maintenant au contenu de la méthode `start()` qui représente le point d'entrée du programme.

La classe [Form](#) permet de créer une page ou l'équivalent d'une fenêtre d'une application desktop. Vous pouvez y placer des composants graphiques comme des boutons et autres. Dans notre code, le constructeur de cette classe prend en paramètre un texte (*Hi World*) qui est le titre de la fenêtre (ou de la page).

❸ Cette ligne permet de créer un [Label](#) affichant le texte *Hi World* (ce composant permettant d'afficher du texte, de l'image ou les deux) et de l'insérer à l'intérieur de notre page avec la méthode `addComponent()` de la classe [Form](#).

❹ Cette dernière ligne permet d'afficher notre page à l'écran.

Place maintenant au contenu de la méthode `stop()` avec l'explication du point ❺. Dans notre classe, un [Form](#) nommé `current` est déclaré en tant que variable d'instance. Cette variable permet de récupérer la page courante (celle qui s'affiche à l'écran). Cette récupération se fait avec la méthode `getCurrent()` qui se trouve à l'intérieur de la classe [Display](#) (qui est un singleton). La question est de savoir pourquoi nous récupérons la page

courante ! Comme mentionné à la [Section 2, Structure d'une application Codename One](#), la méthode `stop()` permet d'effectuer une action avant que l'application ne se mette en pause. Ceci nécessite une petite explication.

Sous des plateformes comme iOS et BlackBerry OS, quand vous réduisez une application pour effectuer une autre action comme recevoir un appel par exemple, votre application est automatiquement mise en pause. Quand vous revenez dessus, vous constaterez que l'application redémarre et ça peut être embêtant si vous étiez en train de faire quelque chose d'important. C'est pour cette raison qu'avant que l'application ne soit mise en pause, le code à l'intérieur de cette méthode récupère la page courante pour la sauvegarder dans une variable. Quand vous reviendrez sur l'application après, l'appareil fera appel de nouveau à la fonction `start()`. Ainsi, le code au début de cette méthode vérifie d'abord si la valeur de `current` est nulle. Si ce n'est pas le cas alors la page sauvegardée est affichée.

4. La compilation avec Codename One

L'une des particularités de Codename One est l'utilisation du cloud pour la compilation. Ce choix a ses avantages et ses inconvénients. Même en utilisant un framework multiplateforme, il est normal d'installer sur son ordinateur les kits de développement natif de chacune des plateformes à cibler. Cela veut dire que dans le cas de Codename One, vous devriez normalement installer cinq SDK avant de pouvoir compiler vos applications pour les cinq plateformes supportées. Heureusement, nous n'avons pas à le faire à cause de la présence du cloud qui a été mise à disposition pour ça.

Note > *Contrairement à ce que pourraient penser certains, ce ne sont pas les codes sources de vos programmes qui sont envoyés sur le serveur de Codename One mais plutôt les bytecodes (les fichiers d'extension .class) générés à partir des classes de vos programmes par le compilateur de Java.*

Avantages de la compilation dans le cloud :

- être épargné de l'installation et des paramétrages des divers SDK natifs sur son ordinateur ;
- ne pas avoir à apprendre à utiliser les émulateurs ou simulateurs fournis avec chaque SDK ;
- ne pas avoir à acheter ou à utiliser un Mac pour compiler pour la plateforme iOS si on est un utilisateur de Windows ou de Linux ;
- ne pas avoir à acheter ou à utiliser un ordinateur Windows pour compiler pour la plateforme BlackBerry et Windows Phone si on est un utilisateur de Mac ou de Linux.

Inconvénients de la compilation dans le cloud :

- ne pas pouvoir compiler sur son propre ordinateur, ce qui serait un gain de temps considérable ;
- la nécessité de faire appel parfois à des outils des SDK natifs pour effectuer certains débogages particuliers.

Note > *L'accès au cloud de Codename One peut être gratuit comme payant. Tout au long de ce livre, nous utiliserons un compte gratuit qui donne accès à la grande majorité des fonctionnalités dont nous aurons besoin. Un compte gratuit inclut 100 crédits de compilation par mois. Chaque compilation (pour Android, Blackberry, Windows phone, J2ME) vous enlèvera 1 crédit sauf la compilation iOS qui enlèvera 8 crédits. Ceci s'explique par le coût élevé de l'hébergement d'un Mac OS dans le cloud. Un tableau contenant la différence entre un compte utilisateur gratuit et un compte utilisateur payant est accessible sur le [site de Codename One](#).*

Avant d'effectuer une compilation, vous devez d'abord vous rendre sur le site officiel de Codename One pour y [créer un compte gratuit](#). Ce compte vous permettra de suivre

l'évolution de vos compilations dans le cloud, de récupérer vos exécutables et d'accéder à d'autres fonctionnalités utiles.

Pour la création d'un compte, cliquez sur le bouton Signup (qu'on peut retrouver sur n'importe quel page du site en haut à droite) pour accéder au formulaire d'inscription. Remplissez ce formulaire. Une fois l'inscription effectuée, identifiez-vous (voir la [Figure 1.8](#)) pour accéder à votre espace membre personnel qui doit ressembler à celui de la [Figure 1.9](#).

Figure 1.8 : Formulaire d'identification

The screenshot shows a 'Login' interface. At the top left is the word 'Login'. On the right is a small 'X' icon. Below the title are two input fields: one for 'E-MAIL:' and one for 'PASSWORD:'. At the bottom are three buttons: 'Password Reset' (gray), 'Cancel' (gray), and 'Login' (blue).

Figure 1.9 : Espace membre

Your Builds appear here. Notice that older builds are automatically deleted to preserve server space!

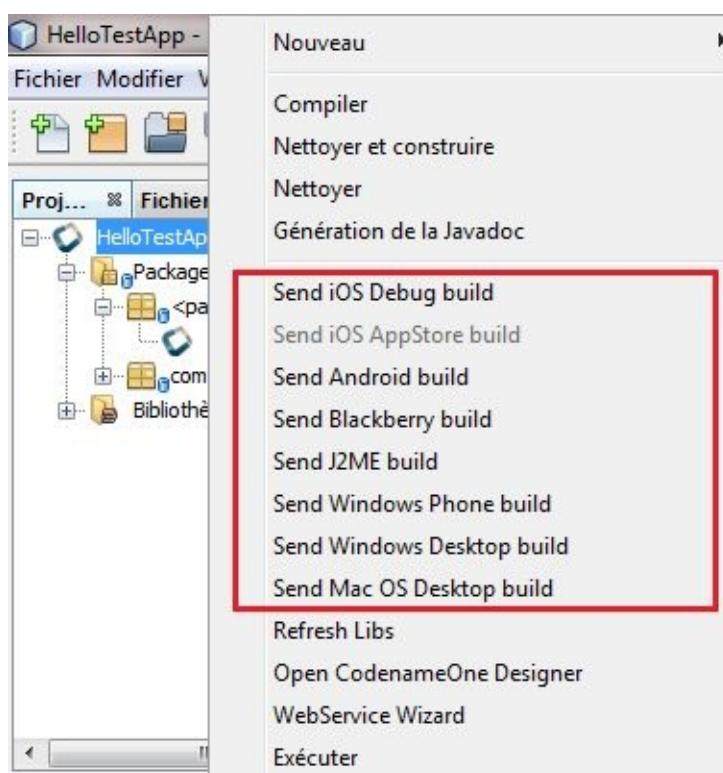
Platform	Build Name	Duration
Android	Successful build - 'TestPlugin'	0:41
Android	Build Error	Mon Feb 23 2015 20:16:03 GMT+0000 (Maroc)
Android	Build Error	Wed Mar 11 2015 13:42:13 GMT+0000 (Maroc)
Android	Build Error	Wed Mar 11 2015 13:45:37 GMT+0000 (Maroc)
Android	Successful build - 'CN1Upload'	0:39
Android	Successful build - 'CN1Upload'	0:38

En fonction de votre type d'abonnement (gratuit ou payant), vous pouvez avoir plus d'onglets (donc plus de fonctionnalités) dans votre espace privé. Pour notre cas d'utilisateur gratuit, nous avons accès aux onglets suivants :

- Build : Sous cet onglet, vous verrez l'état en cours de vos compilations en ligne. Quand la compilation est en cours, il sera marqué *Building* [Compilation en cours]. Si elle réussit, *Successfully build* [Compilation réussie]. Si elle échoue, *Build Error* [Erreur de compilation] et un fichier journal (ou fichier *log*) au format texte contenant l'erreur qui s'est produite vous sera proposé en téléchargement.
- Subscription : Cet onglet permet de choisir un type d'abonnement payant selon vos besoins. Il est possible de souscrire à un essai d'un mois à un compte PRO pour tester les fonctionnalités payantes du cloud (comme l'utilisation des **notifications push**) avant de se décider par la suite. Un abonnement payant vous donnera aussi un accès gratuit à une formation vidéo complète. Cette page vous indique également le nombre de compilations que vous pouvez encore faire dans le mois courant (ceci concerne seulement les utilisateurs du service gratuit qui sont limités à 100 compilations par mois).
- Account : Ici, vous pouvez modifier vos informations personnelles à savoir le nom, l'e-mail, le mot de passe et autres.

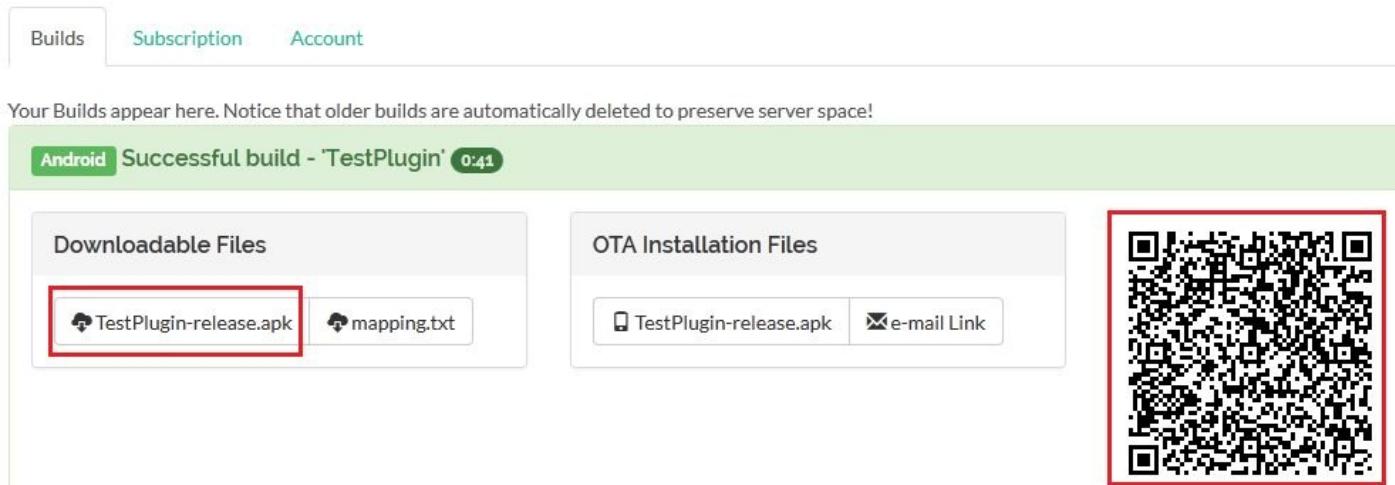
Voyons maintenant comment lancer une compilation en ligne. La méthode suivante concerne les utilisateurs NetBeans. Les utilisateurs d'Eclipse et de IDEA s'y retrouveront facilement. Dans NetBeans, faites un clic droit sur votre projet pour avoir accès au menu contextuel et sélectionnez l'une des entrées encadrées sur la [Figure 1.10](#).

Figure 1.10 : Lancer une compilation en ligne



Le menu comporte des entrées de type Send XXX build, où XXX indique le nom de la plateforme que vous voulez cibler. Ainsi, un clic sur Send Android build par exemple permettra de lancer la compilation de votre application pour la plateforme Android, ce qui va créer un exécutable d'extension .APK. S'il s'agit de votre première compilation alors une fenêtre s'affichera pour vous demander votre identifiant (e-mail) et votre mot de passe. Entrez-les et validez. Une fois la compilation terminée, connectez-vous à votre espace membre en ligne et téléchargez l'exécutable de votre application en cliquant sur le nom de l'exécutable du fichier ou en scannant le QRCode de l'exécutable avec votre téléphone ou tablette (voir la [Figure 1.11](#)).

Figure 1.11 : Accès à l'exécutable des applications compilées dans le cloud

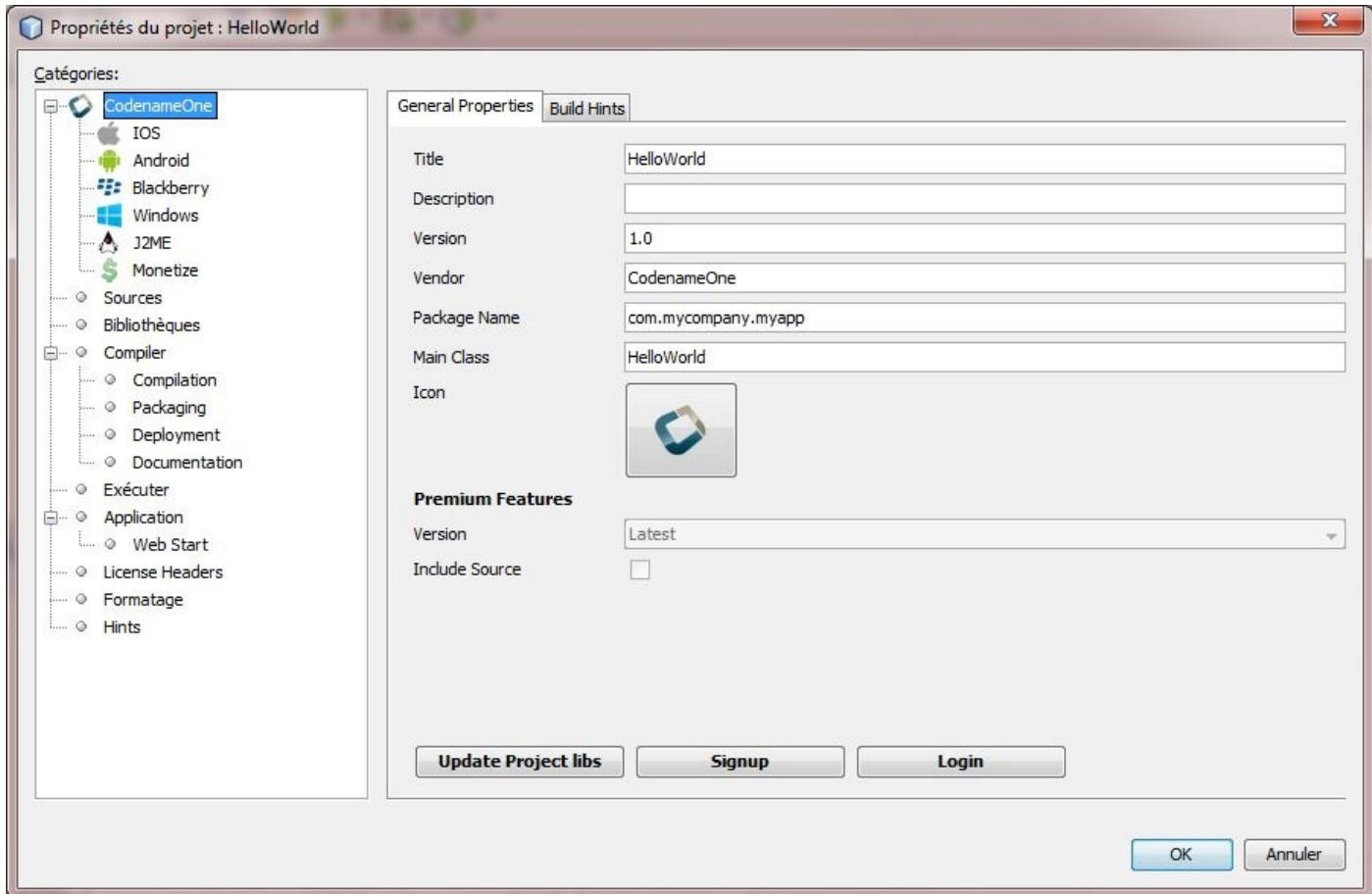


Avant de lancer une compilation de votre application dans le cloud, prenez soin de bien générer et de configurer dans les paramètres de votre projet (voir [Figure 1.12](#)) les fichiers de signature pour la ou les plateformes que vous voulez cibler. L'accès à ces paramètres se fait en cliquant droit sur le nom de votre projet et en sélectionnant l'entrée Propriétés.

Note > Pour plus de détails sur la signature d'une application avec Codename One avant son déploiement, voir l'[Annexe 3 : Signature d'une application](#).

Note > Pour plus de détails sur les arguments de compilation, voir l'[Annexe 2 : Les arguments de compilation](#).

Figure 1.12 : Interface des propriétés d'une application Codename One



5. Présentation et fonctionnement du simulateur

Le fait que Codename One soit fourni avec un simulateur est un grand avantage. Cela permet d'effectuer tous les tests possibles sur l'ordinateur sans être obligé d'installer et de supporter la lenteur des émulateurs de certaines plateformes (émulateurs Android et BlackBerry OS). Étant un simulateur (et non un émulateur), les comportements sur les plateformes ne sont pas forcément reproduits fidèlement, mais c'est le prix à payer pour avoir un outil flexible et s'adaptant au mieux à diverses plateformes.

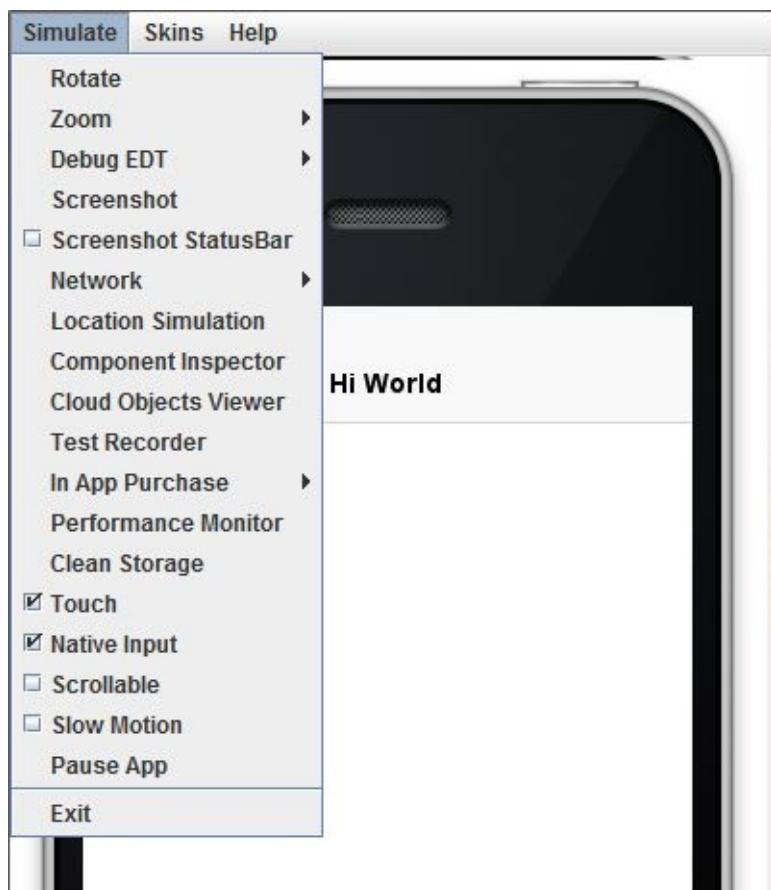
Le simulateur est composé de trois menus : Simulate, Skins et Help.

Le menu Simulate contient tout ce qu'il faut pour effectuer des simulations et interagir aussi avec vos applications.

Le menu Skins fournit des thèmes pour chaque plateforme supportée par Codename One. Cela permet d'avoir une idée du rendu des applications en fonction de chaque plateforme.

Le menu Help propose des raccourcis vers les pages de la documentation, du forum et vers le serveur de compilation.

5.1. Le menu Simulate



Rôle des fonctionnalités et des outils du menu Simulate :

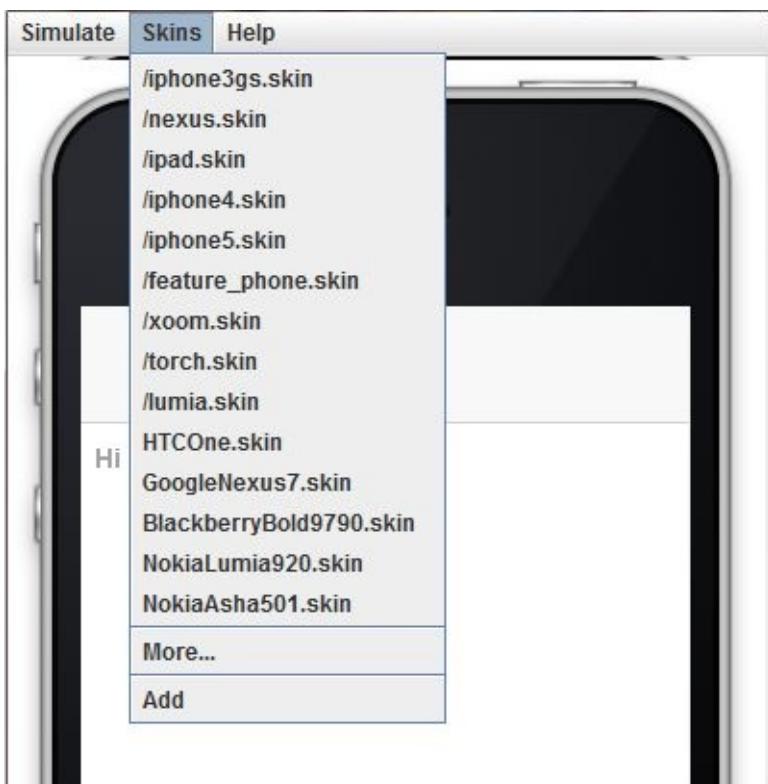
- Rotate : Permet de changer l'orientation du simulateur et de basculer du mode portrait (vertical) au mode paysage (horizontal) et vice versa.
- Zoom : Permet de zoomer l'affichage.
- Debug EDT : Permet de traquer et de savoir s'il y a une violation de l'[EDT \(Event Dispatch Thread\)](#). L'EDT est le thread (processus léger) principal qu'utilise Codename One pour effectuer la majorité de ses tâches (dessins de l'interface, gestion des événements, etc.). Voir [l'annexe 1](#) pour plus de détails.
- Screenshot : Permet d'effectuer une capture d'écran de l'application qui est en cours d'exécution.
- Screenshot StatusBar : Quand cette commande est activée (cochée), la capture d'écran de l'application contiendra la barre de statut. Si elle est décochée, alors cette barre sera juste absente de l'image capturée.
- Network : Cette commande permet de choisir le niveau de la vitesse de la connexion internet à utiliser pour vos tests. Elle fournit aussi un outil nommé [Network monitor](#) pour traquer les requêtes d'envoi et de réception des données à travers le réseau. Cela aide à trouver facilement les problèmes qui pourraient se produire pendant la communication de vos applications avec un serveur distant.
- Location Simulation : Fournit un outil pour simuler la fonctionnalité de géolocalisation.
- Component Inspector : Affiche la hiérarchie des composants graphiques utilisés dans les applications. Vous obtiendrez aussi quelques informations utiles sur ces composants.
- Cloud Objects Viewer : Fournit tout ce qu'il faut pour gérer les enregistrements de données que vous ferez dans le cloud de Codename One. Vous pouvez y charger vos données déposées dans le cloud, effectuer des requêtes, etc.
- Test Recorder : Permet d'effectuer des tests unitaires sans écrire du code.
- Purchase : Permet de simuler la fonctionnalité de paiement à l'intérieur d'une application ([In-App purchase](#)). Simule les achats, les abonnements, les remboursements.
- Performance Monitor : Comme son nom l'indique, cet outil sert à traquer la performance. Il permet de voir les informations sur la vitesse de dessin des composants graphiques qui composent les applications. Vous verrez aussi le nombre de fois où chacun de ces composants a été invoqué dans l'application, etc.
- Clean Storage : Permet de supprimer toutes les données stockées dans les *Storage* (l'une des méthodes de stockage disponible en Codename One).
- Touch : Quand elle est cochée (état par défaut) alors l'écran du simulateur devient tactile et on peut cliquer dessus. Quand elle est décochée, les clics sur l'écran du simulateur n'ont plus d'effets (utile pour les écrans non tactiles comme ceux des plateformes J2ME et de certains BlackBerry).
- Native Input : Cochée (état par défaut), elle permet au simulateur d'utiliser le

système d'entrée de données natif de la plateforme (clavier pour le cas d'un test sur ordinateur). Si elle est décochée alors le simulateur utilisera le clavier virtuel intégré à Codename One pour gérer l'entrée des données.

- Scrollable : Quand ce menu est coché (état par défaut), il n'est pas possible d'agrandir à volonté la taille de la fenêtre du simulateur sans tronquer l'affichage. Par contre, on aura accès à une barre de défilement. En mode décochée, cela est possible mais la barre de défilement sera absente.
- Slow Motion : Permet d'afficher au ralenti les transitions d'une interface à une autre.
- Pause App : Permet de simuler la mise en pause de l'application. Cet état de pause est activé lorsqu'un événement extérieur (un appel, l'arrivée d'un SMS, etc.) intervient pendant l'exécution de l'application.
- Exit : Permet de fermer le simulateur.

Le contenu du menu Simulate change de temps en temps donc il se peut que d'autres fonctionnalités s'y ajoutent après la publication de ce livre.

5.2. Le menu Skins



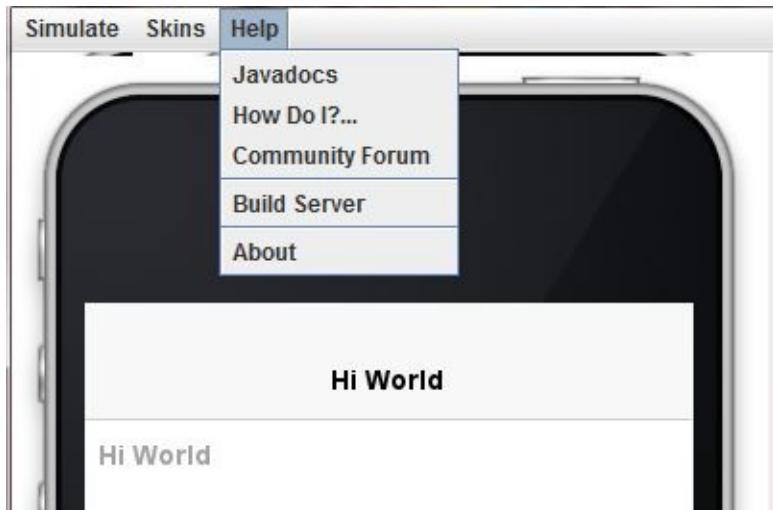
Le menu Skins fournit divers habillages et styles visuels, constitués de thèmes, des images des plateformes, des polices de caractères et des propriétés (type de clavier, taille de police par défaut, police de caractères par défaut, etc.) des plateformes supportées. Chaque plateforme a son style d'affichage. Vous pouvez ainsi avoir un visuel fidèle de vos applications sur chacune de ces plateformes.

- More... : Ce menu ouvre une fenêtre permettant de télécharger et d'installer

automatiquement d'autres skins de plateformes ou de certaines marques de smartphones.

- Add : Permet de choisir et de charger manuellement un skin téléchargé sur l'ordinateur.

5.3. Le menu Help



- Javadocs : Permet d'ouvrir la documentation technique (la Javadoc) de toutes les classes de l'API.
- How Do I?... : Donne accès à une page contenant des tutoriels vidéo.
- Community Forum : Donne accès au forum du site. Il s'agit d'un forum vraiment actif.
- Build Server : Donne accès à la page permettant de s'identifier à l'espace membre du développeur dans le but de récupérer les exécutables des applications compilées.

Les composants graphiques

Créer une interface graphique belle, conviviale, intuitive et personnalisable est tout aussi important que les fonctionnalités d'une application. C'est dans cette logique que nous allons voir dans ce chapitre les principaux composants de l'API graphique de Codename One. Tous ne seront pas abordés mais nous verrons la majorité d'entre eux. Avant toute chose, vous devez savoir que tout composant en Codename One hérite d'une classe nommée Component et certains de ces composants sont plus élaborés que d'autres et peuvent contenir d'autres composants.

Attention > *Les méthodes des différentes classes exposées dans ce chapitre sont loin de constituer la liste complète des méthodes de ces classes. Il s'agit seulement de ceux que vous utiliserez le plus souvent.*

1. Les conteneurs principaux

1.1. Container

Comme l'indique son nom, Container est un composant qui peut contenir d'autres composants. Il n'est pas possible de l'afficher directement, mais certains composants qui héritent de ses méthodes ont la possibilité de s'afficher à l'écran. Le composant Container représenté par la classe du même nom peut contenir trois types d'objets : un ou plusieurs autres Container, un [gestionnaire de positionnement](#) (ou *layout*) et un ou plusieurs [autres composants](#), enfants de la classe mère Component ou de Container lui-même.

Plusieurs composants héritent de la classe Container. Il s'agit par exemple des composants [Form](#), [Dialog](#), [Tabs](#), [MapComponent](#), [Calendar](#), et bien d'autres. Le fait qu'un Container peut contenir un ou plusieurs autres Container permet de construire des interfaces plus complexes et plus abouties.

Création d'un Container

Pour créer un Container, on a le choix entre l'utilisation de l'un des deux constructeurs de la classe :

- `Container()` – Crée un Container ayant par défaut un gestionnaire de positionnement (layout) intégré. Le layout choisi par défaut est le [FlowLayout](#), qui permet d'aligner les composants les uns à la suite des autres sur une même ligne (voir [Section 2, Les layouts ou gestionnaires de positionnement](#) pour plus d'informations sur les layouts).
- `Container(Layout layout)` – Crée un Container avec un layout que vous allez définir vous-même à l'opposé du FlowLayout qui est installé par défaut si aucun layout n'est spécifié.

La majorité des méthodes de la classe Container sont orientées vers l'ajout ou la suppression de composants. Elles permettent aussi de gérer les composants insérés dans le Container et d'obtenir des informations comme le nombre total de composants dans le Container, l'accès à un composant inséré à une position déterminée, etc.

Exemple 2.1 : Code de création d'un Container

```
Container c=new Container() ;  
Container c=new Container(new BoxLayout(BoxLayout.Y_AXIS));
```

Quelques méthodes de Container

- `void addComponent(Component cmp)` – Ajoute un composant à un Container.
- `void addComponent(int index, component)` – Ajoute un composant à une position spécifique du Container. La position est spécifiée en premier paramètre et le second paramètre est le composant à ajouter.
- `void revalidate()` – Rafraîchit le contenu d'un Container avec une réorganisation des éléments du layout intégré.
- `void animateLayout(int duree)` – Cette méthode a le même rôle que `revalidate()` avec en plus un effet visuel d'animation. En paramètre, vous devez spécifier la durée de l'animation en millisecondes.
- `Component getComponentAt(int index)` – Retourne le composant qui se trouve à la position spécifiée en paramètre.
- `int getComponentCount()` – Retourne le nombre total de composants à l'intérieur d'un Container.
- `void removeComponent(Component cmp)` – Enlève un composant spécifique du Container. Mettez en paramètre le composant à enlever.
- `void removeAll()` – Permet d'enlever tous les composants à l'intérieur du Container.
- `void setEnabled(boolean enabled)` – Active et désactive tous les composants insérés à l'intérieur du Container. Le paramètre `enabled` doit prendre la valeur `true` (pour l'activation) ou `false` (pour la désactivation).
- `void setLayout(Layout layout)` – Définit le layout (gestionnaire de positionnement) qui sera intégré à la racine du Container. En paramètre de cette méthode, le layout à intégrer.
- `void setScrollable(boolean scrollable)` – Active et désactive la fonctionnalité de défilement horizontal et vertical du Container. Par défaut, la valeur de son paramètre est à `true`. Cette méthode est l'équivalent de l'appel des méthodes `setScrollableX` et `setScrollableY` avec une valeur `true` pour chacune d'elles.

1.2. Form

Le composant Form est l'un des principaux composants de la bibliothèque graphique et hérite de la classe Container. Il permet de créer une page, un formulaire ou l'équivalent d'une fenêtre pour les applications desktop (il s'agit ici de divers noms pour désigner la même chose). C'est sur ce composant que vous allez ajouter le plus souvent les composants visuels (boutons, listes déroulantes, images, etc.). Contrairement à un Container brut, Form est un Container de haut niveau constitué par défaut de trois zones.

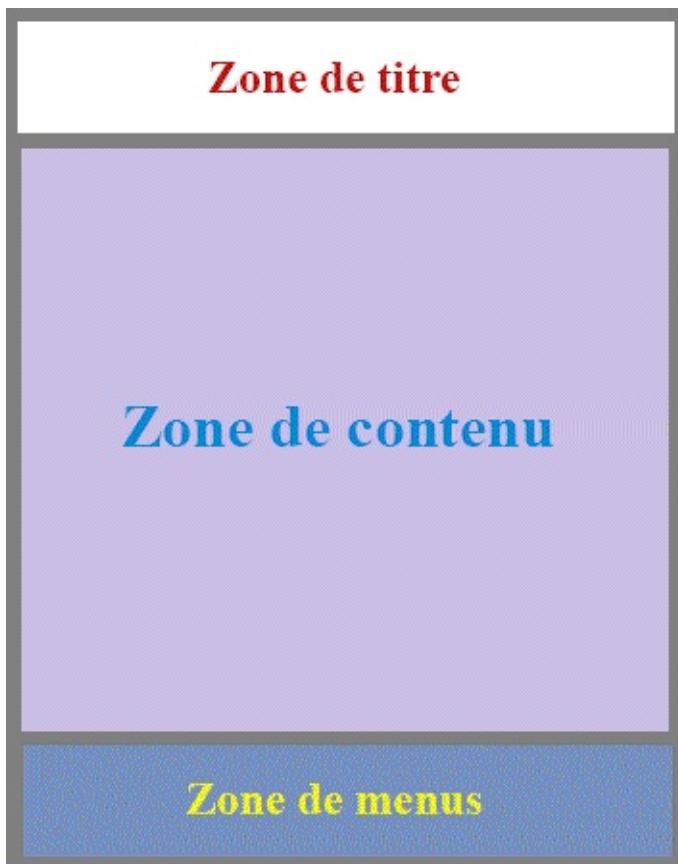
- une zone de titre ;
- une zone de contenu ;

- une zone de menu qu'on peut ne pas utiliser.

Il est possible d'accéder à chacune de ces zones avec les méthodes `getTitleArea()` pour la zone de titre, `getContentPane()` pour la zone de contenu et `getMenuBar()` pour la barre de la zone de menus. La zone de titre contiendra le titre de l'application, la zone de contenu tous les composants visuels de l'application (boutons, zone de texte, images et autres) et la zone de menu optionnelle les menus de l'application. En fonction de la plateforme visée, les menus s'afficheront par défaut sur l'interface ou pas. Voir la [Section 3.1, Command](#) pour plus d'informations sur les menus.

Note > *On ne peut afficher qu'un seul Form à la fois ce qui fait que l'affichage d'une fenêtre créée par Form masque automatiquement celle qui était précédemment à l'écran.*

Figure 2.1 : Les différentes zones d'un Form



Création d'un Form

Pour créer une fenêtre avec Form, vous devez l'instancier en utilisant l'un des deux constructeurs suivants :

- `Form()` – Crée une fenêtre sans titre.
- `Form(String titre)` – Crée une fenêtre avec un titre qui sera affiché dans la zone de titre. Le titre est spécifié en paramètre.

Exemple 2.2 : Code de création d'un Form

```
Form f=new Form() ;  
Form f=new Form("Titre de la fenêtre");
```

Quelques méthodes de Form

Étant donné que la classe Form dérive de la classe Container, il est possible d'utiliser la majorité des méthodes de Container comme les méthodes d'ajout et de suppression de composants.

- `void setTitle(String titre)` – Spécifie le titre de la page (ou de la fenêtre). Titre en paramètre.
- `void show()` – Lance l'affichage du Form.
- `void addCommand(Command cmd)` – Ajoute un menu de type Command au Form.
- `void setBackCommand(Command backCommand)` – Définit le menu (type Command) qui sera appelé à l'appui de la touche retour présente sur le téléphone.
- `void setTransitionInAnimator(Transition transitionInAnim)` – Permet de définir un effet visuel de transition (représenté sous forme d'animation) pour l'affichage du Form entrant.
- `void setTransitionOutAnimator(Transition transitionOutAnim)` – Permet de définir un effet visuel de transition (représenté sous forme d'animation) avant l'affichage d'un autre Form (ou pour le Form sortant).

1.3. Dialog

La classe Dialog permet de créer l'équivalent d'une boîte de dialogue semblable à ce qu'on peut voir dans une application de bureau. Elle permet de présenter une information à l'utilisateur et n'occupe qu'une partie de l'écran. Elle peut être modale (état par défaut) ou non modale. Quand elle est modale, elle bloque l'accès à l'interface et aux autres fonctionnalités de l'application jusqu'à ce qu'elle soit fermée. Ce blocage affecte aussi l'EDT (le thread principal de Codename One) dont j'ai parlé [plus tôt](#) au chapitre [Démarrage](#).

Dialog contient plusieurs méthodes statiques permettant de créer rapidement une boîte de dialogue modale, mais il est aussi possible de l'instancier soi-même et d'y ajouter des données comme on veut. Cette classe hérite de la classe Form et lui ressemble un peu. On peut pareillement lui ajouter des Command et l'afficher avec la méthode `show()`. À la différence d'un Form, un Dialog n'occupe pas la totalité de l'écran et a une taille plus petite. Son affichage peut être configuré de sorte à ce qu'il reste à l'écran un temps

déterminé (en millisecondes) avant de se fermer. Vous pouvez aussi ne pas spécifier ce temps de fermeture et prévoir un bouton de fermeture qui sera déclenché par l'utilisateur.

Création d'un Dialog

Les deux constructeurs suivants (semblables à ceux de la classe Form) permettent de créer une boîte de dialogue et d'y ajouter soi-même du contenu.

- `Dialog()` – Crée une boîte de dialogue sans titre. Il est possible d'ajouter après un titre avec la méthode `setTitle()`.
- `Dialog(String titre)` – Crée une boîte de dialogue avec un titre.

Quelques méthodes de Dialog

Étant donné que Dialog hérite de Form, il est possible d'utiliser la majorité de ses méthodes.

- `Command showDialog()` – Affiche une boîte de dialogue modale au milieu de l'écran avec un bouton de validation qui servira à fermer la fenêtre du Dialog. Cette validation retourne une valeur de type Command que l'on peut utiliser pour déclencher une action quelconque.
- `void show()` – Affiche une boîte de dialogue modale au milieu de l'écran.
- `void showModeless()` – Affiche une boîte de dialogue non modale.
- `static boolean show(String titre, String texte, String okText, String annulerTexte)` – Cette méthode statique est une méthode surchargée permettant de créer et d'afficher une boîte de dialogue avec du texte sans instancier la classe Dialog. Le premier paramètre est le titre du Dialog, la deuxième est le texte à afficher à l'utilisateur, la troisième est le texte du bouton de validation et le dernier paramètre est le texte du bouton d'annulation. Si l'utilisateur choisit le bouton de validation (paramètre 3) alors cette méthode retournera la valeur booléenne `true`. S'il choisit le bouton d'annulation, elle retournera `false`.
- `void setTimeout(long time)` – Spécifie le temps (en millisecondes) pendant lequel la boîte de dialogue restera affichée à l'écran avant de se fermer automatiquement sans intervention de l'utilisateur.
- `void dispose()` – Ferme la boîte de dialogue. Cette fermeture débloque aussi l'EDT (si la boîte de dialogue est modale) et l'application devient réactive à nouveau.
- `void setDisposeWhenPointerOutOfBounds(boolean dispose)` – Ferme la boîte de dialogue si l'utilisateur pointe à l'écran sur une zone en dehors du contenu de la boîte de dialogue.

- void setDialogType(int dialogType) – Définit le type de la boîte de dialogue. Le paramètre dialogType peut prendre l'une des quatre valeurs suivantes : Dialog.TYPE_CONFIRMATION (pour une confirmation d'action de l'utilisateur), Dialog.TYPE_ERROR (pour signaler une erreur), Dialog.TYPE_WARNING (pour un avertissement), Dialog.TYPE_INFO (pour afficher une simple information), Dialog.TYPE_ALARM (pour une alarme).
- void setDialogPosition(String dialogPosition) – Définit la position d'affichage de la boîte de dialogue à l'écran. Les valeurs possibles du paramètre de position sont les suivantes : BorderLayout.CENTER, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.NORTH, BorderLayout.SOUTH.

Exemple : Boîtes de dialogue sans instantiation manuelle de la classe Dialog

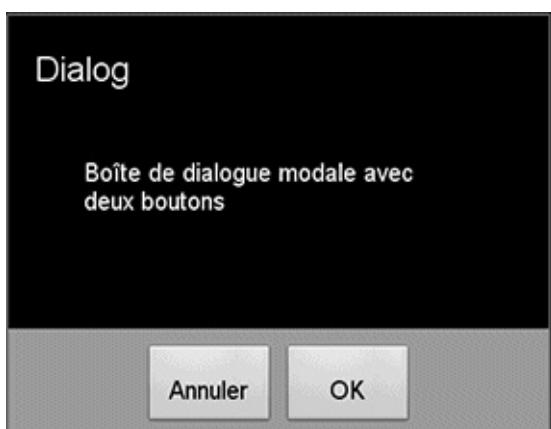
```
Dialog.show("Dialog", "Boîte de dialogue modale avec un bouton", "OK",  
null);
```

donnera :



```
Dialog.show("Dialog", "Boîte de dialogue modale avec deux boutons", "OK",  
"Annuler");
```

donnera :



Exemple : Boîte de dialogue personnalisée avec instantiation

```
Dialog d=new Dialog( "Coucou" );
```

```
❶ d.setTimeout(5000); ❷ Image img=null; ❸ try { img=Image.createImage("/logo.png");  
} catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK", null); } ❹  
Label l=new Label(img); l.setText("Une boîte de dialogue personnalisée"); ❺  
l.setTextPosition(Label.BOTTOM); d.addComponent(l); d.show();
```

❶ Création d'une boîte de dialogue avec une instanciation de Dialog.

❷ Définition du temps d'affichage de la boîte de dialogue avant fermeture.

❸ Chargement d'une image nommée logo.png avec la classe Image.

❹ Crédit à l'auteur de l'application pour l'image utilisée.

❺ Définition de la position du texte du Label par rapport à l'image.

Figure 2.2 : Une boîte de dialogue personnalisée sous iOS 8



Figure 2.3 : Une boîte de dialogue personnalisée sous Android



1.4. Tabs

La classe `Tabs` permet de créer une interface avec des onglets. La position et l'apparence par défaut de ces onglets varient en fonction de la plateforme. Ils peuvent ainsi être disposés en haut, en bas, à gauche ou à droite de l'écran. Sous iOS par exemple, ils sont placés par défaut en bas et sous Android, en haut. Un onglet peut avoir un titre et/ou une image sous forme d'icône. On ne peut ajouter qu'un seul composant à la page d'un onglet. Ainsi, pour avoir plusieurs éléments à l'intérieur de celle-ci, vous devez d'abord placer un Container vide puis vous ajoutez autant de composants visuels que vous voulez dans ce Container.

Note > *L'indice des onglets se compte à partir de la position 0 comme c'est le cas des indices de tableau en programmation. Ainsi, le premier onglet se trouve à l'indice 0, le deuxième à l'indice 1 et le Nième onglet à l'indice N-1.*

Création d'un Tabs

Deux constructeurs pour créer une interface à onglets.

- `Tabs()` – Crée une interface à onglets avec une position par défaut des onglets variant

en fonction de la plateforme.

- `Tabs(int position)` – Crée une interface à onglets en spécifiant la position par défaut en paramètre. Ce paramètre peut prendre les valeurs suivantes : Tabs.TOP (haut), Tabs.BOTTOM (bas), Tabs.LEFT (gauche) et Tabs.RIGHT (droit).

Quelques méthodes de Tabs

- `void addTab(String titre, Component composant)` – Crée une page d'onglet avec un titre en premier paramètre et le composant à placer sur la page en second paramètre.
- `void addTab(String titre, Image icone, Component composant)` – Crée une page d'onglet avec un titre en premier paramètre, une icône en deuxième paramètre et le composant à placer sur la page en troisième paramètre.
- `void insertTab(String titre, Image icone, Component composant, int index)` – Insère une page d'onglet à une position spécifique. Cette position est précisée par le quatrième paramètre.
- `void removeTabAt(int index)` – Supprime un onglet à une position spécifique. Cette position est passée en paramètre.
- `void setSelectedIndex(int index)` – Définit l'onglet actif. La position de cet onglet est précisée en paramètre.
- `int getSelectedIndex()` – Retourne le numéro d'index de l'onglet sélectionné.
- `int getTabCount()` – Retourne le nombre total d'onglets.
- `void setTabPlacement(int tabPlacement)` – Définit la position des onglets à l'écran. Le paramètre tabPlacement peut prendre les valeurs suivantes : Tabs.TOP (haut), Tabs.BOTTOM (bas), Tabs.LEFT (gauche) et Tabs.RIGHT (droit).
- `void setTabTextPosition(int textPosition)` – Définit la position du titre de l'onglet par rapport à l'icône s'il y en a. Les valeurs possibles du paramètre textPosition sont : Component.TOP (haut), Component.BOTTOM (bas), Component.LEFT (gauche) et Component.RIGHT (droit).
- `void setSwipeActivated(boolean activerSwipe)` – Permet de définir si l'utilisateur peut passer d'un onglet à un autre en glissant sur l'écran son doigt de droite à gauche ou vice versa. Cette fonctionnalité est activée par défaut. Mettez comme valeur de paramètre false pour la désactiver et true pour l'activer.
- `Component getTabComponentAt(int index)` – Récupère le composant placé à une position spécifique. Entrez la position en paramètre.
- `void hideTabs()` – Masque les onglets.
- `void showTabs()` – Affiche les onglets s'ils étaient masqués.
- `void addSelectionListener(SelectionListener listener)` – S'abonne à un

listener pour détecter le changement de l'onglet courant. La classe contenant les onglets doit implémenter l'interface SelectionListener qui n'a qu'une seule méthode : void selectionChanged(int oldSelected, int newSelected).

Exemple : Interface avec onglets

```
public class TabsTest implements SelectionListener{

    public void init(Object context) {
        try {
            Resources theme = Resources.openLayered("/theme");
            UIManager.getInstance().setThemeProps(
                theme.getTheme(theme.getThemeResourceNames()[0]));
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    public void start() {

        Form f = new Form("Les onglets");
        f.setLayout(new BorderLayout());

        Tabs t=new Tabs();

        ① Image icone=null; try { icone=Image.createImage("/info.png"); } catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK", null); } ② Container c=new Container(new BoxLayout(BoxLayout.Y_AXIS)); Button identification=new Button("Identification"); Button inscription=new Button("Inscription"); ③ c.addComponent(identification); c.addComponent(inscription); t.addTab("Onglet 1", new Label("Coucou Codename One")); ④ t.addTab("Onglet 2", new TextArea()); ⑤ t.addTab("Onglet 3", icone, c); ⑥ t.addSelectionListener(this);
        f.addComponent(BorderLayout.CENTER,t); ⑦ f.show(); } public void stop() { } public void destroy() { } public void selectionChanged(int oldSelected, int newSelected) {
        Dialog.show("Onglets", "Ancien onglet: "+(oldSelected+1)+" Onglet sélectionné: "+
        (newSelected+1), "Ok", null); ⑧ } }
```

❶ Création de l'interface d'onglets.
❷ Création d'un Container avec deux boutons.
❸ Ajout des deux boutons créés au Container.
❹ Création du premier onglet avec comme composant un **Label** affichant le texte "Coucou Codename One".
❺ Création du deuxième onglet avec comme composant principal une **zone de texte**

➅ multiple.

➆ Création du troisième onglet avec comme composant le Container contenant les deux boutons.

➇ Ajout du Tabs au Form.

➈ Affichage du numéro de l'ancien onglet et du nouvel onglet sélectionné.

Les [Figure 2.4](#), [Figure 2.5](#) et [Figure 2.6](#) montrent le résultat de l'exécution du code de l'exemple.

Figure 2.4 : Contenu du premier onglet (sous iOS)

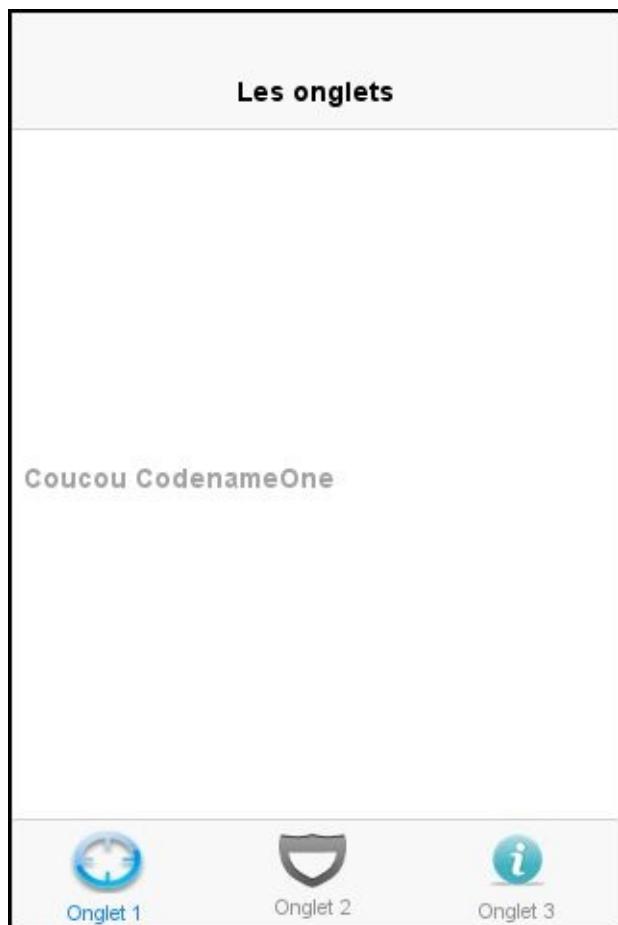


Figure 2.5 : Contenu du deuxième onglet (sous iOS)

Les onglets

Je suis dans une zone de texte multiple



Onglet 1



Onglet 2



Onglet 3

Figure 2.6 : Contenu du troisième onglet (sous iOS)

Les onglets

[Identification](#)

[Inscription](#)



Onglet 1



Onglet 2



Onglet 3

Figure 2.7 : Aperçu des onglets sous Android 2.xx



Figure 2.8 : Aperçu des onglets sous Android 4.x



Un clic sur l'un des onglets affichera ceci :

Figure 2.9 : Affichage du numéro de l'ancien et du nouvel onglet

Les onglets

Onglets

Ancien onglet: 3 Onglet sélectionné: 1

Ok



Onglet 1



Onglet 2



Onglet 3

2. Les layouts ou gestionnaires de positionnement

Les gestionnaires de positionnement, appelés communément *layouts managers* en anglais ou *layouts* tout court, permettent de gérer la disposition des composants dans un Container. En outre, leur utilisation permet aussi de créer des interfaces qui s'adaptent automatiquement à toutes les résolutions d'écran. Ainsi, vous pouvez créer des applications sans vous soucier de la taille et résolution des écrans sur lesquels elles s'exécuteront.

Codename One a neuf layouts mais trois ne sont presque jamais utilisés, donc nous n'allons voir dans cette section que les six principaux. Ces neuf classes héritent d'une classe principale nommée Layout et il est possible de l'éteindre soi-même dans le but de créer son propre layout. Les layouts que nous allons examiner sont les suivants :

- [BoxLayout](#) ;
- [BorderLayout](#) ;
- [FlowLayout](#) ;
- [GridLayout](#) ;
- [LayeredLayout](#) ;
- [CoordinateLayout](#).

Chacun d'eux dispose les composants de manière différente dans un Container. Vous pouvez les combiner pour créer des dispositions plus complexes sur une même interface. C'est ce que nous faisons notamment à la [section Exemple : Contenus fluides avec différents alignements](#).

2.1. BoxLayout

Ce layout permet de disposer de manière verticale ou horizontale les éléments dans un Container.

Création d'un BoxLayout

BoxLayout ne dispose que d'un seul constructeur.

- `BoxLayout(int orientationAxe)` – Crée un layout d'orientation verticale ou horizontale. L'orientation doit être spécifiée en paramètre. Les valeurs possibles de ce paramètre sont `X_AXIS` (pour une orientation horizontale) et `Y_AXIS` (pour une

orientation verticale).

Quelques méthodes de BoxLayout

- `int getAxis()` – Retourne l'axe x et y du layout.
- `void layoutContainer(Container parent)` – Permet d'appliquer le layout aux composants enfants du Container passés en paramètre. Cette méthode est héritée de la classe mère Layout donc elle est accessible pour tous les autres layouts que nous allons voir.

Encadré : Rappel

Chaque classe héritant de Container contient une méthode `setLayout()` permettant d'installer le layout sur le conteneur.

Exemple : Positionner à la verticale ou à l'horizontale

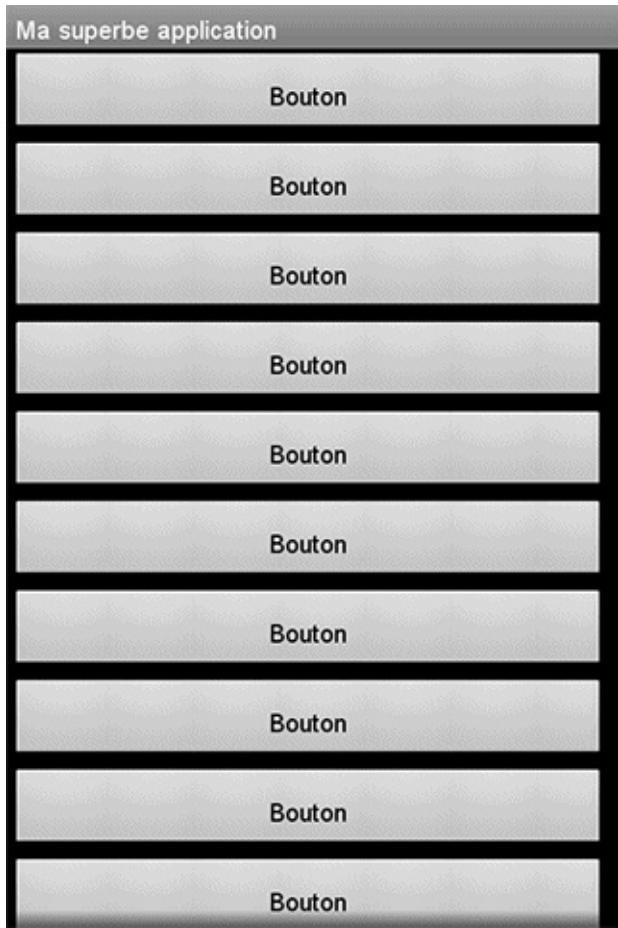
```
public void start() {  
    Form f = new Form("Ma superbe application");  
    BoxLayout layout=new BoxLayout(BoxLayout.Y_AXIS);
```

❶ `f.setLayout(layout);` ❷ `for(int i=0;i<10;i++){ Button button=new Button("Bouton"); f.addComponent(button); }` `f.show();`

❶ Création d'un layout vertical.

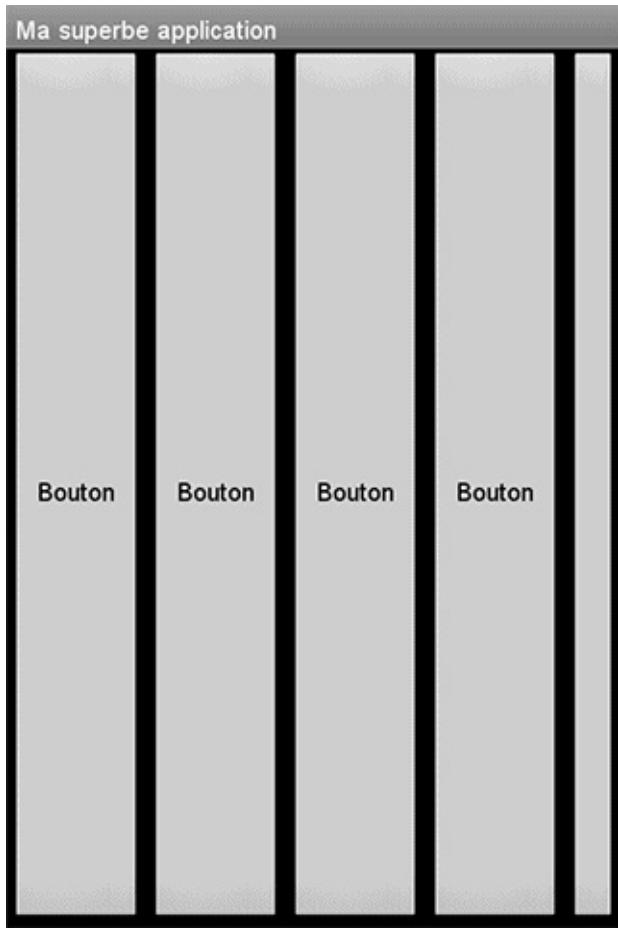
❷ Création de dix boutons suivie de leur ajout sur la fenêtre Form.

Figure 2.10 : Aperçu de la disposition d'un BoxLayout vertical



Remplacez maintenant le `BoxLayout.Y_AXIS` par le `BoxLayout.X_AXIS` et vous obtiendrez ceci :

Figure 2.11 : Aperçu de la disposition d'un `BoxLayout` horizontal



2.2. BorderLayout

BorderLayout permet de disposer les éléments d'un conteneur en se basant sur le concept des quatre points cardinaux. Les éléments devront alors être placés au nord, au sud, à l'est, à l'ouest et au centre du conteneur.

Création d'un BorderLayout

- `BorderLayout()` – Crée un BorderLayout.

Quelques méthodes de BorderLayout

- `Component getCenter()` – Retourne le composant placé au milieu du layout.
- `Component getEast()` – Retourne le composant placé à l'est (à droite) du layout.
- `Component getWest()` – Retourne le composant placé à l'ouest (à gauche) du layout.
- `Component getNorth()` – Retourne le composant placé au nord (au-dessus) du

layout.

- Component getSouth() – Retourne le composant placé au sud (au-dessous) du layout.
- void setCenterBehavior(int centerBehavior) – Définit le comportement du composant placé au milieu du layout. Le paramètre centerBehavior peut prendre l'une des valeurs suivantes :
BorderLayout.CENTER_BEHAVIOR_CENTER,
BorderLayout.CENTER_BEHAVIOR_CENTER_ABSOLUTE,
BorderLayout.CENTER_BEHAVIOR_SCALE,
BorderLayout.CENTER_BEHAVIOR_TOTAL_BELOW.
- void setScaleEdges(boolean scaleEdges) – Permet d'élargir ou d'étendre les composants placés sur les bords du layout (Nord, Sud, Est, Ouest). Une valeur true en paramètre permet de l'activer.

Note > Il n'est pas nécessaire d'avoir un composant à placer sur chacune des cinq zones pour utiliser ce layout. Vous pouvez par exemple ne placer qu'un composant au centre (ou à un autre endroit) et ne pas utiliser les autres zones.

Exemple : Positionner au centre et à la périphérie

```
public void start() {  
  
    Form f = new Form("Ma superbe application");  
    BorderLayout layout=new BorderLayout();
```

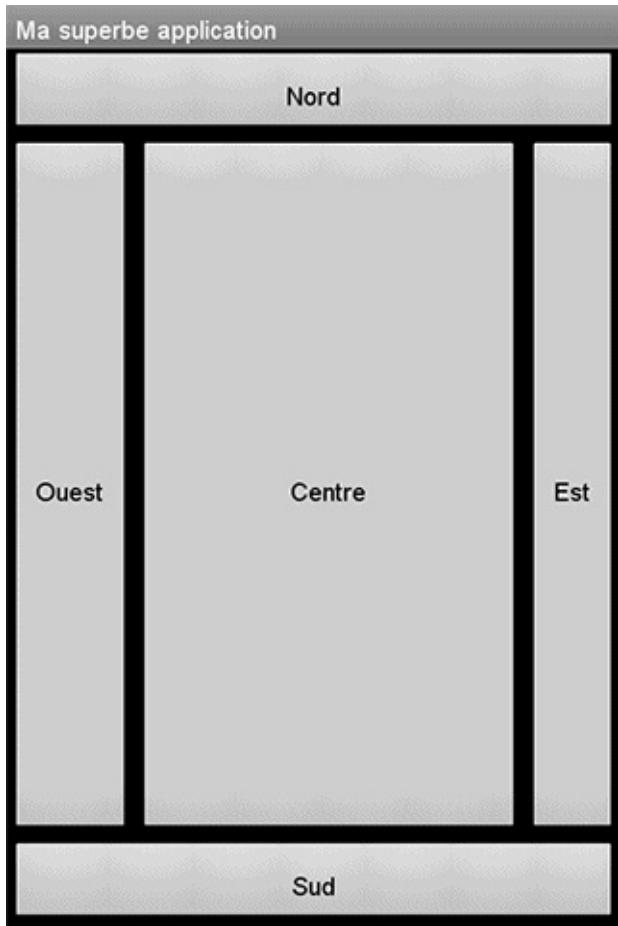
```
❶ f.setLayout(layout); ❷ Button b1=new Button("Nord"); Button b2=new  
Button("Sud"); Button b3=new Button("Est"); Button b4=new Button("Ouest"); Button  
b5=new Button("Centre"); ❸ f.addComponent(BorderLayout.NORTH, b1);  
f.addComponent(BorderLayout.SOUTH, b2); f.addComponent(BorderLayout.EAST, b3);  
f.addComponent(BorderLayout.WEST, b4); f.addComponent(BorderLayout.CENTER,  
b5); f.show(); }
```

❶ Création du layout.

❷ Création de cinq boutons à placer dans les cinq zones du BorderLayout.

❸ Ajout des boutons dans les cinq zones du BorderLayout.

Figure 2.12 : Aperçu de la disposition d'un BorderLayout



2.3. FlowLayout

FlowLayout permet d'aligner les composants sur une ligne (de la gauche vers la droite par défaut) les uns à la suite des autres. S'il n'y a plus assez d'espace sur la ligne, l'alignement continue automatiquement sur la ligne suivante. Par défaut, l'alignement commence à partir de la gauche, mais il est possible de le changer pour qu'il commence à partir de la droite ou du centre.

Création d'un FlowLayout

- `FlowLayout()` – Crée un FlowLayout aligné sur la gauche.
- `FlowLayout(int orientation)` – Crée un FlowLayout avec un alignement spécifié en paramètre. La variable `orientation` peut prendre l'une des valeurs suivantes : `Component.LEFT` (aligné à gauche), `Component.RIGHT` (aligné à droite), `Component.CENTER` (centré).

Quelques méthodes de FlowLayout

- `void setAlign(int orientation)` – Définit l'orientation de l'alignement horizontal. Valeur par défaut à `LEFT`.
- `int getAlign()` – Retourne la valeur de l'orientation de l'alignement horizontal.
- `void setValign(int valign)` – Définit l'orientation de l'alignement vertical à l'intérieur du layout. Les valeurs possibles du paramètre `valign` sont `Component.TOP` (aligné en haut), `Component.BOTTOM` (aligné en bas), `Component.CENTER` (centré).
- `int getValign()` – Retourne la valeur de l'orientation de l'alignement vertical.
- `void setFillRows(boolean fillRows)` – Indique si le layout doit remplir ou non tout l'espace disponible sur la ligne.

Exemple : Contenus fluides avec différents alignements

```
public void start() {
    Form f = new Form("Ma superbe application");
    f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
```

❶ Container `c1=new Container(new FlowLayout());` ❷ Container `c2=new Container(new FlowLayout(Component.RIGHT));` ❸ Container `c3=new Container(new FlowLayout(Component.CENTER));` ❹ ❺ `for(int i=0;i<4;i++){ Button b=new Button("Eric"); c1.addComponent(b); }` ❻ `for(int i=0;i<4;i++){ Button b=new Button("Dodji"); c2.addComponent(b); }` ❽ `for(int i=0;i<4;i++){ Button b=new Button("Patricia"); c3.addComponent(b); }` ❾ `f.addComponent(c1); f.addComponent(c2); f.addComponent(c3); f.show(); }`

❶ Création d'un layout vertical. Il sera le layout principal de l'interface et est placé dans Form.

❷ Création d'un premier Container et d'un FlowLayout aligné à gauche. Nous pouvons être plus précis en passant `Component.LEFT` en paramètre au constructeur de FlowLayout.

❸ Création d'un deuxième Container et d'un FlowLayout aligné à droite.

❹ Création d'un troisième Container et d'un FlowLayout centré.

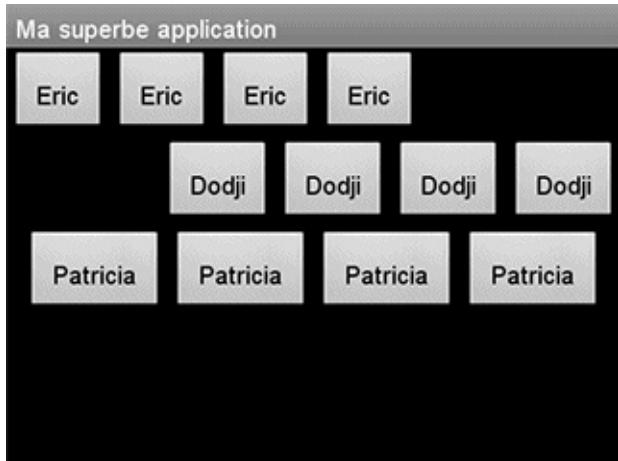
❺ Création de quatre boutons et insertion dans le premier Container.

❻ Création de quatre boutons et insertion dans le deuxième Container.

❼ Création de quatre boutons et insertion dans le troisième Container.

❽ Ajout des trois Container dans le Form. Ces trois Container seront alignés dans le layout vertical de Form.

Figure 2.13 : Aperçu de la disposition d'un FlowLayout



2.4. GridLayout

Ce layout permet de disposer les composants dans les cellules (de tailles identiques) d'une grille invisible. Pour le créer, vous devez spécifier le nombre de lignes et de colonnes de la grille et imaginer le résultat comme un tableau à remplir avec des composants.

Création d'un GridLayout

- `GridLayout(int ligne, int colonne)` – Crée un GridLayout avec les nombres de lignes et de colonnes indiqués en paramètres.

Quelques méthodes de GridLayout

- `int getColumns()` – Retourne le nombre de colonnes dans le layout.
- `int getRows()` – Retourne le nombre de lignes dans le layout.
- `void setAutoFit(boolean autoFit)` – Permet aux colonnes et aux lignes de s'adapter automatiquement à l'espace disponible sur l'interface de l'application quand le paramètre est à true.
- `void setFillLastRow(boolean fillLastRow)` – Permet de remplir complètement la dernière ligne du layout si le nombre d'éléments sur cette ligne est plus élevé. Il suffit de passer la valeur du paramètre à true pour le faire.

Exemple : Positionner selon une grille

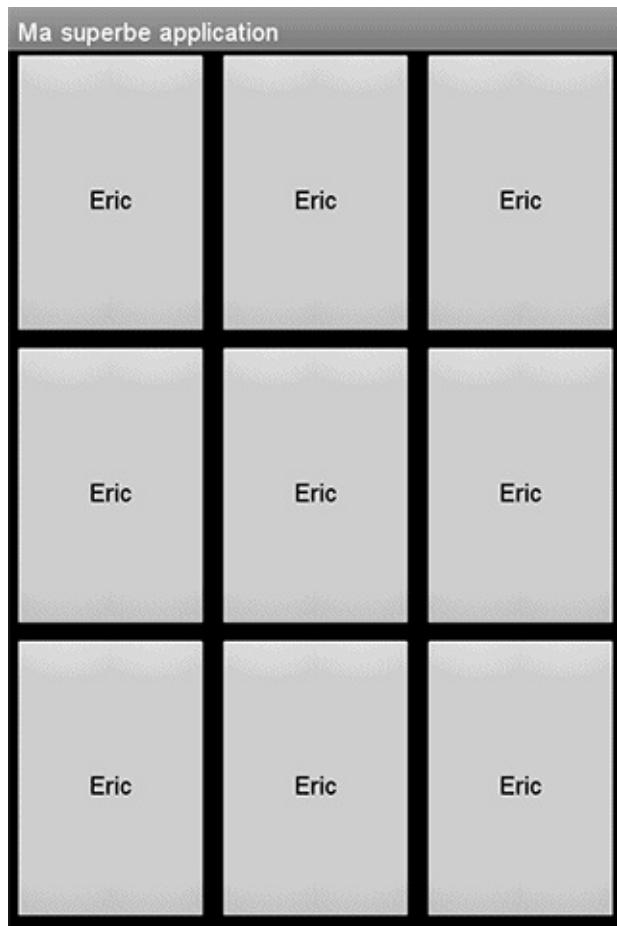
```
public void start() {  
    Form f = new Form("Ma superbe application");  
    GridLayout layout=new GridLayout(3,3);
```

❶ f.setLayout(layout); ❷ for(int i=0;i<9;i++){ Button b=new Button("Eric");
f.addComponent(b); } f.show(); }

❶ Création d'un layout composé de trois lignes et de trois colonnes.

❷ Création de neuf boutons et insertion dans le Form.

Figure 2.14 : Aperçu de la disposition d'un GridLayout



2.5. LayeredLayout

Ce layout un peu particulier permet de placer un composant au-dessus d'un autre (effet de superposition) sans rien faire d'autre de spécial.

Création d'un LayeredLayout

- LayeredLayout() – Crée un LayeredLayout .

À part les méthodes héritées de la classe mère Layout, ce layout n'a pas d'autres méthodes particulières.

2.6. CoordinateLayout

À la différence des layouts précédemment vus, qui requièrent de placer les composants dans un ordre bien déterminé, CoordinatorLayout permet de les positionner à n'importe quel endroit de l'écran en utilisant les coordonnées X et Y. En outre, comme avec LayeredLayout, vous pouvez superposer des composants. Le layout les place les uns au-dessus des autres, ce qui crée un effet de chevauchement s'ils sont proches. Ainsi, le dernier composant ajouté est celui qui est au-dessus.

Création d'un CoordinatorLayout

- CoordinatorLayout() – Crée un CoordinatorLayout sans la possibilité d'élargissement du layout.
- CoordinatorLayout(int largeur, int hauteur) – Crée un CoordinatorLayout avec une taille déterminée. En paramètre, vous spécifiez la largeur et la hauteur que devra occuper le layout à l'intérieur du conteneur dans lequel il sera placé.
- CoordinatorLayout(Dimension d) – Ce constructeur est identique au précédent à la différence près que vous utiliserez la classe Dimension pour spécifier la taille du layout. Dimension a aussi un constructeur qui prend en paramètres la largeur et la hauteur.

Avec CoordinatorLayout, vous devez utiliser les méthodes setX() et setY() des composants à placer dans le conteneur pour effectuer les positionnements à l'écran. L'exemple qui suit montre comment le faire.

À part les méthodes héritées de la classe mère Layout, ce layout n'a pas d'autres méthodes particulières.

Exemple : Positionner avec des coordonnées

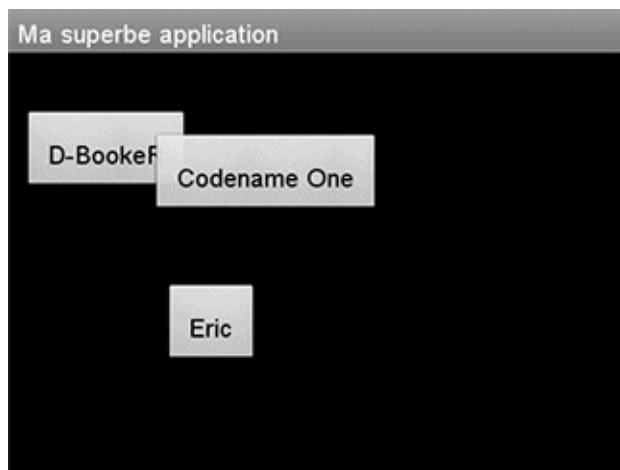
```
public void start() {  
    Form f = new Form("Ma superbe application");  
    CoordinatorLayout layout = new CoordinatorLayout(f.getWidth(),  
f.getHeight());  
  
    ❶ f.setLayout(layout); ❷ Button b1=new Button("D-BookeR"); Button b2=new  
    Button("Codename One"); Button b3=new Button("Eric"); ❸ b1.setX(10); b1.setY(50);  
    b2.setX(110); b2.setY(70); b3.setX(120); b3.setY(200); f.addComponent(b1);  
    f.addComponent(b2); f.addComponent(b3); f.show(); }
```

Création du layout ayant pour dimensions la largeur et la hauteur du Form. Les ❶ méthodes `getWidth()` et `getHeight()` permettent de récupérer respectivement la valeur de la largeur et de la hauteur du Form.

❷ Création de trois boutons.

❸ Positionnement des trois boutons sur l'interface avec les méthodes `setX()` et `setY()` permettant d'attribuer respectivement la valeur de la position X et de la position Y.

Figure 2.15 : Aperçu de la disposition d'un `CoordinateLayout`



3. Autres composants

Dans cette section, nous allons passer en revue différents composants dont vous aurez besoin pour concevoir une interface graphique complète.

3.1. Command

Command est un composant qui permet de créer un menu. L'apparence et l'emplacement d'un menu varient en fonction des plateformes. Sur certaines, il apparaîtra directement sur l'interface (cas de l'iOS) et sur d'autres, son affichage sera activé en appuyant sur une touche dédiée (physique ou disponible sur l'interface de l'application) du téléphone (cas d'Android et de Blackberry).

Une fois qu'un menu est créé avec la classe Command, il faut le rattacher à l'interface d'une fenêtre (un Form) ou à une boîte de dialogue (Dialog). Par défaut, un menu contient du texte. On peut aussi lui ajouter une icône. Pour créer un menu, vous pouvez utiliser l'un des quatre constructeurs ci-dessous.

- `Command(String command)` – Crée un menu avec en paramètre le mot ou texte à afficher sur le menu.
- `Command(String command, Image icone)` – Crée un menu avec en premier paramètre le mot ou texte à afficher sur le menu et en second paramètre l'image de l'icône du menu.
- `Command(String command, Image icone, int id)` – Crée un menu avec en premier paramètre le mot ou texte à afficher sur le menu, en second paramètre l'image de l'icône du menu et en dernier paramètre un identifiant unique pour désigner le menu.
- `Command(String command, int id)` – Crée un menu avec en paramètre le mot ou texte à afficher sur le menu et en second paramètre un identifiant unique pour désigner le menu.

Exemple : Page avec titre et menus

Voici un exemple d'une interface graphique contenant un titre et deux menus. L'un permet d'afficher une boîte de dialogue et l'autre de fermer l'application. Avant de passer à l'exemple, créez un nouveau projet et dans le dossier `src` de votre projet, placez l'image d'une petite icône comme celle-ci  que je vais utiliser. Vous trouverez cette icône dans les sources du livre.

```
public class FormEtMenus implements ActionListener {  
  
    private static int APPROPOS=0;  
    private static int QUITTER=1;
```

```

public void init(Object context) {
    try {
        Resources theme = Resources.openLayered("/theme");
        UIManager.getInstance().setThemeProps(
            theme.getTheme(theme.getThemeResourceNames()[0]));
    } catch(IOException e){
        e.printStackTrace();
    }
}

public void start() {
    Form f = new Form("Les menus");
}

```

❶❷ Image icone=null; **try** { icone=Image.createImage("info.png"); } **catch** (IOException ex) { Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null); } Command apropos=**new** Command("A propos",icone,APROPOS); **❸** Command quit=**new** Command("Quitter",QUITTER); **❹❺** f.addCommand(apropos); f.addCommand(quit); **❻** f.addCommandListener(**this**); **❼** f.show(); } **public void** stop() { } **public void** destroy() { } **public void** actionPerformed(ActionEvent evt) { Command cmd=evt.getCommand(); **❽** int cId=cmd.getId(); **❾** **if**(cId==APROPOS){ **❿** Dialog.show("Hi!", "Salut! J'utilise Codename One", "OK", null); **❻** } **else** **if**(cId==QUITTER){ Display.getInstance().exitApplication(); **❽** } } }

❶ Création d'une fenêtre avec un titre.

❷ Chargement de l'image de l'icône qui sera placée sur le menu A propos.

❸ Création d'un menu A propos avec une icône.

❹ Création d'un menu Quitter.

❺ Ajout des menus à la fenêtre créée par Form.

❻ Abonnement du Form à l'interface ActionListener représentée ici par notre classe FormEtMenus.

❼ Affichage de Form.

❽ Récupération du menu ayant déclenché l'événement.

❾ Récupération de l'identifiant unique du menu sélectionné.

❿ Test pour déterminer l'identifiant du menu récupéré.

❻ Si l'identifiant récupéré correspond à celui du premier menu A propos alors afficher une boîte de dialogue avec un message.

② Si l'identifiant récupéré correspond à celui du second menu Quitter alors fermer l'application en utilisant la méthode `exitApplication()` de la classe `Display`.

Dans cet exemple, j'ai utilisé un identifiant unique (`id`) pour récupérer et reconnaître le menu qui a envoyé le signal d'événement. Une autre méthode serait de déclarer nos `Command` en tant que variables d'instance de la classe et dans la méthode `actionPerformed`. Le code serait ceci :

```
Command cmd=evt.getCommand();
if(cmd==apropos){
    Dialog.show("Hi!", "Salut! J'utilise Codename One", "OK", null);
} else if(cmd==quit){
    Display.getInstance().exitApplication();
}
```

Une troisième alternative serait de récupérer le nom de la `Command` avec `getCommandName()` et d'effectuer le test sur ce nom :

```
Command cmd=evt.getCommand();
String nomCmd=cmd.getCommandName();

if(nomCmd.equals("A propos")){
    Dialog.show("Hi!", "Salut! J'utilise Codename One", "OK", null);
} else if(nomCmd.equals("Quitter")){
    Display.getInstance().exitApplication();
}
```

À vous de choisir la manière que vous préférez ou celle avec laquelle vous êtes le plus à l'aise. La [Figure 2.16](#) montre un aperçu de l'exécution du code de notre exemple dans le simulateur.

Figure 2.16 : Interface créée avec Form comportant deux menus (sous iOS)

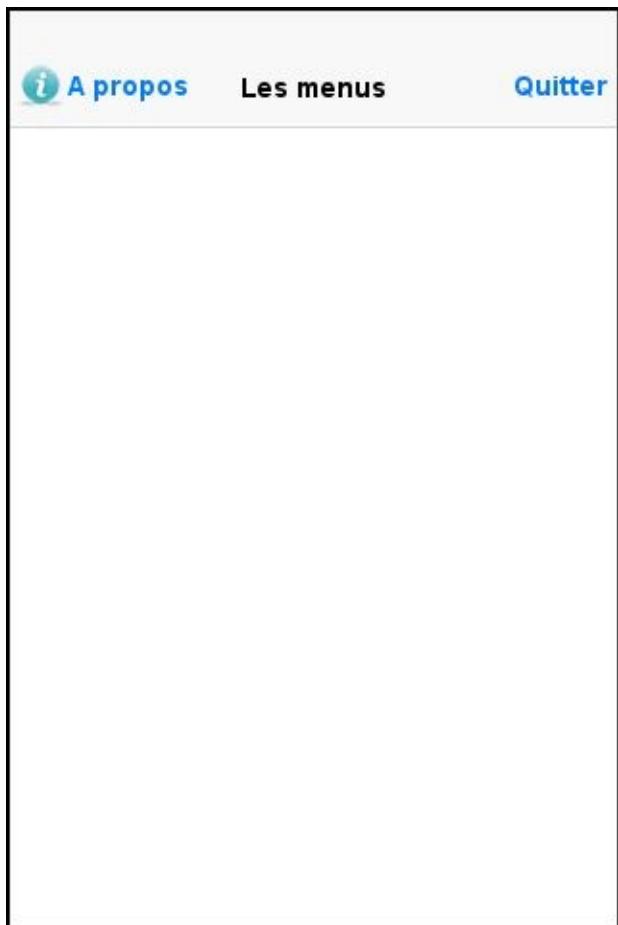
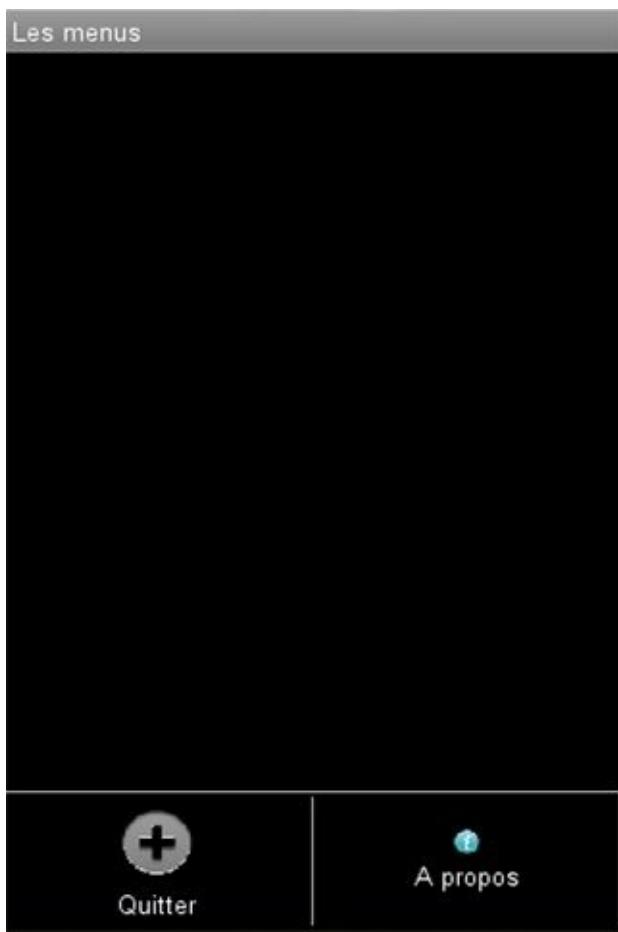


Figure 2.17 : Interface créée avec Form comportant deux menus (sous Android)



Un clic sur le menu A propos affichera une boîte de dialogue avec un message comme le montrent les [Figure 2.18](#) et [Figure 2.19](#).

Figure 2.18 : Résultat du clic sur le menu A propos – gestion d'un événement (sous iOS 8)

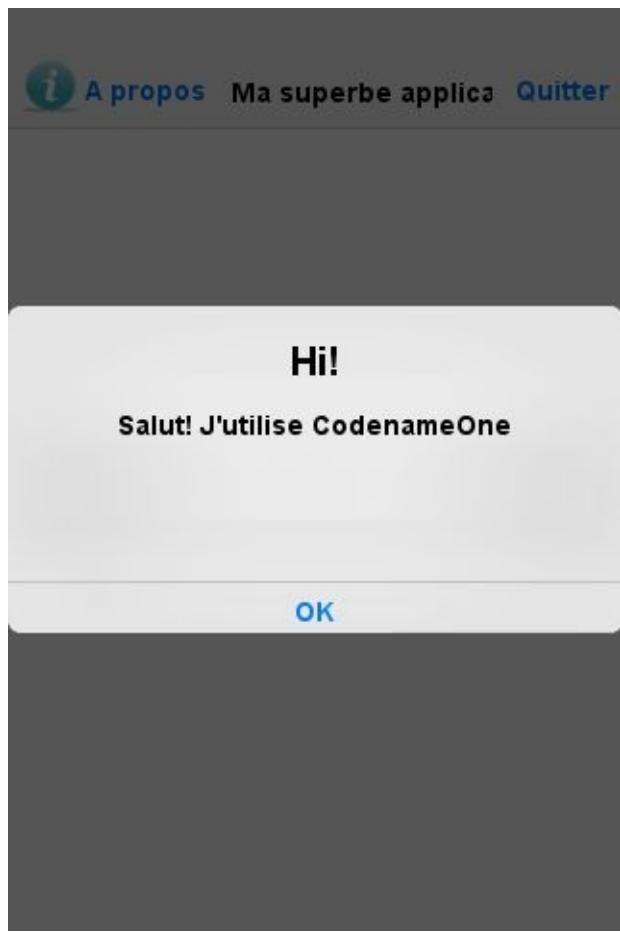


Figure 2.19 : Résultat du clic sur le menu A propos – gestion d'un événement (sous Android)



Note > La gestion des événements des menus doit être effectuée à l'intérieur de la méthode `actionPerformed(ActionEvent)` héritée de l'interface `ActionListener`. Dans notre exemple, c'est notre classe qui a hérité de cette interface et qui a implémenté cette méthode.

Quelques méthodes de Command

- `void setIcon(Image icone)` – Attribue une icône au menu si cela n'a pas été fait avec le constructeur.
- `void setCommandName(String nom)` – Attribue un nom au menu.
- `void getCommandName()` – Récupère le nom d'un menu.
- `void setEnabled(boolean etat)` – Active/Désactive un menu.
- `int getId()` – Récupère l'identifiant unique d'un menu.

3.2. Label

Le composant Label permet d'afficher du texte et/ou de l'image. Le texte affiché par Label n'occupe qu'une ligne et sera tronqué s'il est plus long que la dimension de l'écran. Ainsi,

le texte va défiler pour permettre à l'utilisateur de le lire si ce cas se présente. Étant donné qu'un Label peut avoir à la fois une image et du texte, il est possible de définir la position de ce texte par rapport à l'image.

Exemple : Afficher un texte court ou une image

```
❶ Label text=new Label(); text.setText("Je suis un joli label"); ❷ Image icone=null; try { icone=Image.createImage("/bonhomme.jpg"); } catch (IOException ex) { Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null); } Label img=new Label(icone); ❸ Label imgtxt=new Label(icone); imgtxt.setText("Hey Einstein"); imgtxt.setTextPosition(Component.RIGHT);
```

Création d'un simple label avec un texte à afficher. La méthode `setText()` permet ❶ d'ajouter du texte à un label. Vous pouvez aussi l'utiliser pour modifier le texte existant.

Pour afficher une image avec la classe Label, il faut d'abord charger cette image avec la classe Image puis la passer en paramètre au constructeur de Label.

On peut faire la même chose en utilisant la méthode `setIcon(Image img)` au lieu de passer directement la variable de l'image au constructeur de Label. L'image à charger peut être déposée à trois endroits différents : dans le dossier SRC de votre projet, dans un dossier quelconque à l'intérieur du dossier SRC ou encore à l'intérieur du fichier de ❷ ressources .RES de votre projet. Nous reviendrons sur ce troisième cas plus loin dans l'ouvrage au chapitre [Codename One Designer](#).

Note > *Une erreur de chargement de l'image lèvera une exception donc n'oubliez pas de récupérer l'erreur en question dans la console ou en l'affichant par exemple via une boîte de dialogue comme effectué dans l'exemple.*

Attention > *N'oubliez pas le caractère slash (/) qui précède le nom du fichier de l'image à charger.*

Dans cet exemple, la méthode `setTextPosition()` est utilisée pour placer le texte à droite de l'image du Label. Les valeurs possibles de cette méthode sont : ❸ Component.TOP (haut), Component.BOTTOM (bas), Component.LEFT (gauche), Component.RIGHT (droit).

Figure 2.20 : Aperçu d'un Label avec texte et/ou image



Quelques méthodes de Label

- `void setTickerEnabled(boolean tickerEnabled)` – Permet d'activer ou de désactiver le défilement du label à l'écran si celui est trop long.
- `void startTicker()` – Permet de déclencher le défilement du texte.
- `void startTicker(long delay, boolean rightToLeft)` – Cette méthode est semblable à la précédente et offre en plus la possibilité de définir soi-même quelques paramètres. Le premier paramètre `delay` permet de spécifier en millisecondes le temps entre deux défilements. Le second paramètre `rightToLeft` permet de définir la direction de défilement du texte. Par défaut, le défilement se passe de la droite vers la gauche. En passant la valeur de ce paramètre à `false`, le défilement se fera alors dans le sens inverse.
- `void stopTicker()` – Stoppe un défilement.
- `void setIcon(Image img)` – Attribue une icône (image) à un label.

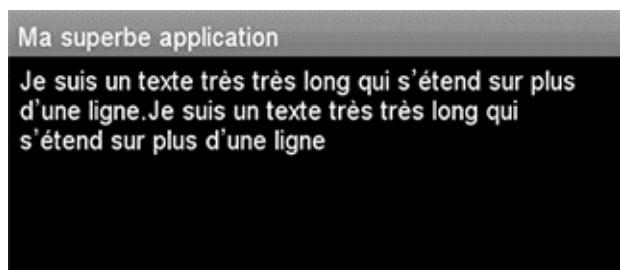
3.3. SpanLabel

SpanLabel est un Label qui peut contenir un long texte écrit sur plusieurs lignes. Même s'il est très proche de Label, il n'hérite pas de ce dernier, mais directement de Container. Il contient aussi les méthodes `setText(String text)` et `setIcon(Image img)`.

Exemple : Afficher un texte long

```
SpanLabel text=new SpanLabel("Je suis un texte très très long qui s'étend  
sur plus d'une ligne."  
+"Je suis un texte très très long qui s'étend sur plus d'une ligne.");
```

donnera :



3.4. Button

Pour créer un bouton, il faut utiliser la classe `Button`. Cette classe hérite de `Label` et permet aussi d'afficher du texte et/ou de l'image. Ce composant peut recevoir des événements de clic. Un bouton peut avoir divers états : l'état par défaut, l'état de l'appui du bouton, l'état du bouton quand il a le focus.

Exemple : Créer un bouton simple

```
❶ Button b1=new Button("Premier bouton"); ❷ Image icone=null; try {  
    icone=Image.createImage("/bonhomme.jpg"); } catch (IOException ex) {  
    Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null); } Button b2=new  
    Button(icone); ❸ Button b3=new Button("Troisième bouton", icone);
```

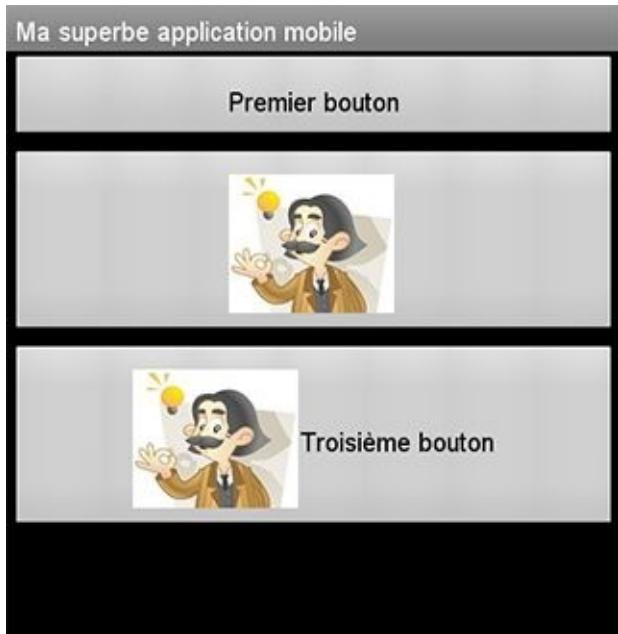
❶ Crédit d'un bouton simple avec du texte.

Chargement de l'image à ajouter au bouton suivi de la création d'un bouton contenant l'image chargée.

❷ **Attention** > N'oubliez pas le caractère slash (/) qui précède le nom du fichier de l'image à charger.

❸ Crédit d'un troisième bouton contenant une image et du texte.

Figure 2.21 : Aperçu d'un bouton avec texte et/ou image



Gestion de clic sur un bouton

Pour gérer les événements de clic sur un bouton, il faut faire appel à l'interface ActionListener. Le bouton doit s'abonner à cette interface et traiter l'action à exécuter dans sa méthode actionPerformed(ActionEvent). L'exemple suivant affiche un bouton contenant un texte qui change après un clic sur le bouton.

```
final Button b1=new Button("J'adore Mary J. Blige");
b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        b1.setText("J'adore aussi Iggy Azalea");
    }
});
```

Figure 2.22 : Avant le clic

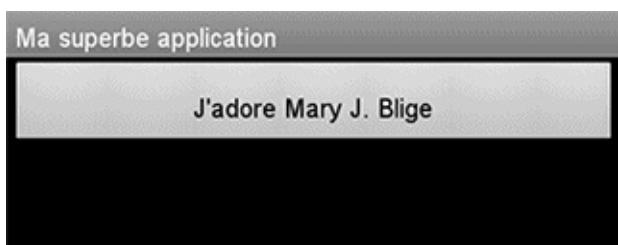
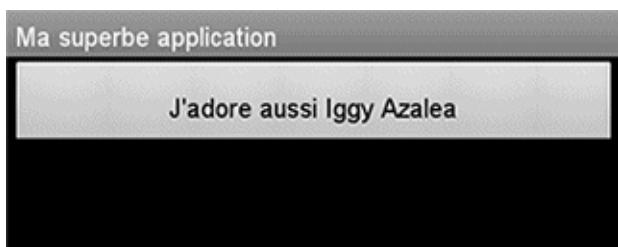


Figure 2.23 : Après le clic



Note > Tous les composants qui héritent de Button (et d'autres composants aussi) utilisent aussi l'interface ActionListener pour la gestion des événements.

Quelques méthodes de Button

- `int getState()` – Retourne l'état courant du bouton. La valeur renvoyée sera l'une des trois suivantes : STATE_DEFAULT (état par défaut quand le bouton n'est pas appuyé ou n'a pas le focus), STATE_PRESSED (état du bouton quand il est appuyé), STATE_ROLLOVER (état du bouton quand il a le focus).
- `void pressed()` – Change l'état du bouton en mode appuyé (pendant le clic).
- `void released()` – Change l'état du bouton en mode relâché (après le clic).
- `void setCommand(Command cmd)` – Intègre au bouton un menu créé avec la classe Command. Le bouton réagira alors de la même manière que le menu.
- `void setPressedIcon(Image pressedIcon)` – Définit l'image qui sera affichée quand le bouton sera pressé.
- `void setRolloverIcon(Image rolloverIcon)` – Définit l'image qui sera affichée quand le bouton aura le focus.

3.5. SpanButton

Dans la même logique du rôle que joue SpanLabel à côté d'un Label, SpanButton permet de créer un bouton qui peut contenir un long texte sur plusieurs lignes. Il hérite de Container et donne aussi un accès aux méthodes `setText(String text)`, `setIcon(Image img)`, `setIconPosition(String position)`, `getText()`, `getIcon()`, `getIconPosition()`.

Exemple : Bouton avec un texte long

```
SpanButton sb=new SpanButton("Je suis un texte très très long sur un  
bouton et je m'étends sur plus d'une ligne."  
+"Je suis un texte très très long sur un bouton et je m'étends sur plus  
d'une ligne.");
```

donnera :

Je suis un texte très très long sur un bouton et je m'étends sur plus d'une ligne. Je suis un texte très très long sur un bouton et je m'étends sur plus d'une ligne.

3.6. MultiButton

MultiButton est un bouton au visuel un peu particulier. Ce bouton peut afficher jusqu'à quatre lignes de texte en plus d'une icône et d'un emblème (représenté généralement par une image ou par une case à cocher). L'affichage d'un emblème est optionnel. MultiButton peut être utilisé comme un bouton, un label, une case à cocher unique ou multiple. La gestion de clic sur un MultiButton se fait de la même manière que celui d'un Button.

Exemple : Bouton complexe

```
public void start() {
    Form f = new Form("Frameworks mobiles multiplateformes");
    BoxLayout layout= new BoxLayout(BoxLayout.Y_AXIS);
    f.setLayout(layout);

    Image icone=null;
    Image embleme=null;
    try {
        icone=Image.createImage("/cn1logo.png");
        embleme=Image.createImage("/arrow.png");
    } catch (IOException ex) {
        Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null);
    }

    MultiButton mb1=new MultiButton();
    mb1.setTextLine1("CodenameOne");
    mb1.setTextLine2("SDK utilisant Java");
    mb1.setTextLine3("Formidable et unique");
    mb1.setTextLine4("www.codenameone.com");
    mb1.setIcon(icone);
    mb1.setEmblem(embleme);
    mb1.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent evt) {
            Dialog.show("Codename One", "Créez votre prochain succès avec
codename One", "Ok", null);
        }
    });
}
```

```

});

MultiButton mb2=new MultiButton();
mb2.setRadioButton(true);
mb2.setSelected(true);
mb2.setTextLine1("Xamarin");
mb2.setTextLine2("SDK utilisant le C#");

MultiButton mb3=new MultiButton();
mb3.setCheckBox(true);
mb3.setSelected(true);
mb3.setTextLine1("Titanium");
mb3.setTextLine2("SDK utilisant le JavaScript");

MultiButton mb4=new MultiButton();
mb4.setHorizontalLayout(true);
mb4.setTextLine1("PhoneGap");
mb4.setTextLine2("HTML/JS/CSS");
mb4.setTextLine3("Adapté aux développeurs web");

f.addComponent(mb1);
f.addComponent(mb2);
f.addComponent(mb3);
f.addComponent(mb4);

f.show();
}

```

Les méthodes `setTextLineX()` permettent d'écrire un texte sur les lignes du `MultiButton`. X représente une valeur allant de un à quatre pour chacune des quatre lignes. Ainsi, `setTextLine1()` permet d'écrire sur la première ligne du bouton, `setTextLine2()` sur la deuxième ligne, `setTextLine3()` sur la troisième ligne, `setTextLine4()` sur la quatrième ligne.

Figure 2.24 : Différents types de `MultiButton` (iOS 8)

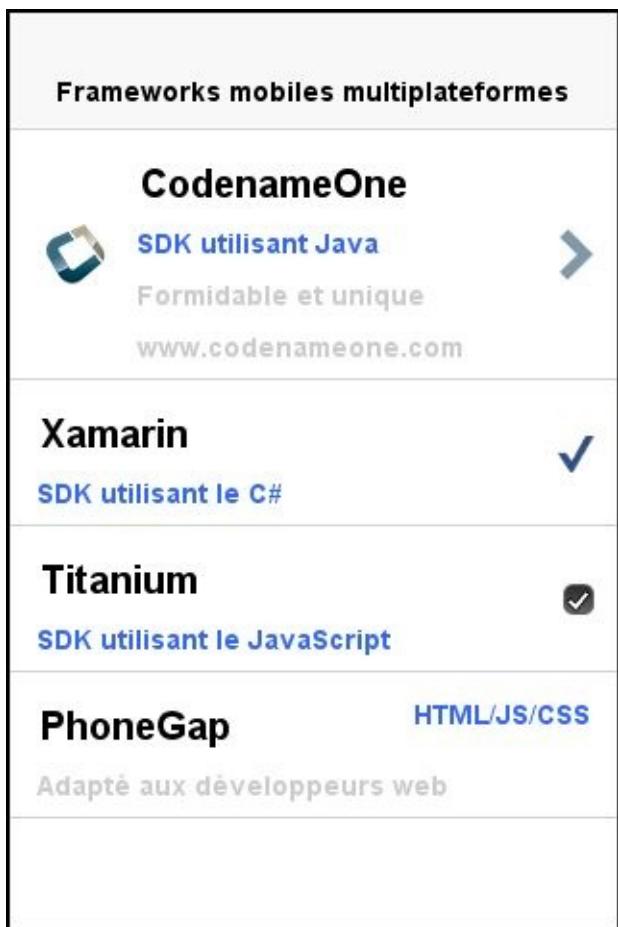
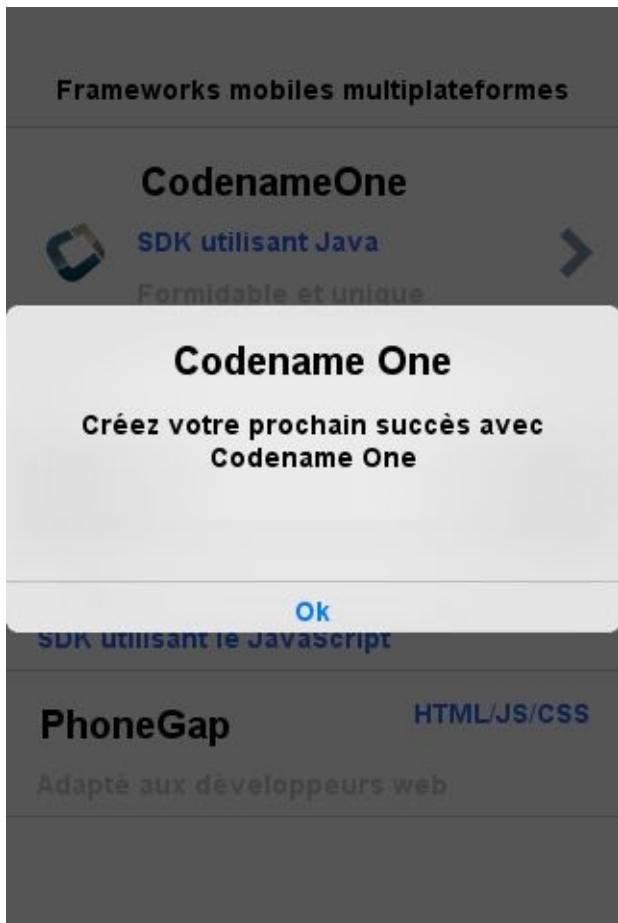


Figure 2.25 : Après un clic sur le premier MultiButton



Quelques méthodes de MultiButton

- `void setEmblem(Image em)` – Ajoute une image d’emblème au MultiButton. Le paramètre `em` représente l’image en question.
- `void setEmblemPosition(String position)` – Définit la position d’un emblème. Cette position est inspirée des zones de disposition que BorderLayout propose. Ainsi, les valeurs possibles du paramètre `position` sont : `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.EAST`, `BorderLayout.WEST`.
- `void setRadioButton(boolean b)` – Permet de transformer le MultiButton en une case à cocher à choix unique (communément appelée un **RadioButton**) tout en gardant les avantages d’un MultiButton. Il suffit de passer la valeur du paramètre à `true` pour obtenir cela.
- `void setGroup(String group)` – Indique le groupe du bouton si le Multibutton est transformé en RadioButton avec `setRadioButton()`.
- `void setCheckBox(boolean b)` – Permet de transformer le MultiButton en une case à cocher à choix multiple (communément appelée **CheckBox**) tout en gardant les avantages d’un MultiButton. Passez la valeur du paramètre à `true` pour obtenir cela.
- `void setSelected(boolean b)` – Permet de cocher ou de décocher le bouton s’il est traité comme une case à cocher.
- `boolean isSelected()` – Retourne l’état du bouton (coché ou décoché) s’il est traité comme une case à cocher.
- `void setHorizontalLayout(boolean b)` – Permet d’aligner les deux premiers textes du bouton côté à côté quand le paramètre est fixé à `true`. Le second texte (de la seconde ligne) sera alors placé à l’emplacement qu’occupe par défaut l’image de l’emblème.
- `void setInvertFirstTwoEntries(boolean b)` – Inverse l’ordre des deux premières lignes de texte. Le texte de la première ligne occupera celui de la seconde ligne et celui de la seconde ligne occupera celui de la première.

3.7. CheckBox

CheckBox est un composant qui permet de créer une case à cocher à choix multiple. Il dérive de Button et peut donc utiliser les méthodes de cette classe. Un CheckBox a deux états à savoir l’état coché et l’état décoché. Il peut prendre du texte, de l’image ou les deux comme c’est le cas d’un Label ou d’un Button. L’apparence d’un CheckBox varie en fonction de la plateforme.

Tout comme un bouton normal, un CheckBox peut recevoir des événements de clic qui peuvent être traités aussi par l’interface ActionListener.

Exemple : Cases à cocher avec texte et/ou image

```
Image imageAnd=null;
Image imageIos=null;
try {
    imageAnd=Image.createImage("/android.png");
    imageIos=Image.createImage("/ios.png");
} catch (IOException ex) {
    Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null);
}

CheckBox checkb=new CheckBox("Codename One");

❶ CheckBox checkb2=new CheckBox(imageAnd); ❷ CheckBox checkb3=new
CheckBox("iPhone, iPad.", imageIos); ❸
```

- ❶ Crée un CheckBox composé uniquement d'un texte.
- ❷ Crée un CheckBox composé uniquement d'une image.
- ❸ Crée un CheckBox composé d'un texte et d'une image.

Figure 2.26 : Aperçu d'un CheckBox avec texte et/ou image



Exemple : Gérer le clic sur une case à cocher

L'exemple suivant affiche une boîte de dialogue indiquant si la case est cochée ou décochée après un clic sur le CheckBox.

```
final CheckBox checkb=new CheckBox("D-BookeR");
checkb.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {
    if(checkb.isSelected()){
        Dialog.show("Etat bouton", "Cette case est cochée", "OK", null);
    }
}}
```

```
    } else {
        Dialog.show("Etat bouton", "Cette case est décochée", "OK", null);
    }
});
```

Figure 2.27 : Case cochée



Figure 2.28 : Case décochée



Quelques méthodes de CheckBox

- `void setSelected(boolean selected)` – Permet de cocher/décocher un CheckBox.
- `boolean isSelected()` – Retourne l'état (coché ou décoché) d'un CheckBox. `true` si c'est coché et `false` dans le cas contraire.

3.8. RadioButton

Un RadioButton permet de créer des cases à cocher n'autorisant qu'un choix à la fois contrairement au CheckBox qui en autorise plusieurs. Tout comme un CheckBox, un RadioButton peut être composé de texte, d'image ou des deux. RadioButton s'utilise en général avec un composant nommé ButtonGroup. Ce composant est une sorte de conteneur invisible qui permet de gérer l'état (coché ou décoché) de plusieurs RadioButton regroupés ensemble à l'intérieur d'un même ButtonGroup. Ainsi, grâce à ce composant, on est assuré d'avoir un seul RadioButton coché à la fois. Par défaut, les RadioButton dans un ButtonGroup sont décochés.

Exemple : Cases à cocher à choix unique

```

Image cn1=null;
Image xamarin=null;
try {
    cn1=Image.createImage("/cn1logo.png");
    xamarin=Image.createImage("/xamarin.png");
} catch (IOException ex) {
    Dialog.show("Erreur de chargement", ex.getMessage(), "OK", null);
}

ButtonGroup group=new ButtonGroup();
RadioButton rb1=new RadioButton("Codename One", cn1);

```

❶ RadioButton rb3=new RadioButton(xamarin); ❷ RadioButton rb2=new RadioButton("Titanium"); ❸ ❹ group.add(rb1); group.add(rb2); group.add(rb3);

- ❶ Crée un RadioButton composé d'un texte et d'une image.
- ❷ Crée un RadioButton composé uniquement d'une image.
- ❸ Crée un RadioButton composé uniquement d'un texte.
- ❹ Ajout des RadioButton au ButtonGroup. L'ajout se fait avec la méthode add().

Figure 2.29 : Sous Android 4.x



Attention > Même si les RadioButton ont été ajoutés au ButtonGroup, ce n'est pas le ButtonGroup qui doit être ajouté au final dans votre conteneur principal (Form par exemple) mais plutôt les RadioButton.

Quelques méthodes de RadioButton

- void setSelected(boolean selected) – Permet de cocher/décocher un CheckBox.
- boolean isSelected() – Retourne l'état (coché ou décoché) d'un CheckBox. true si c'est coché et false au cas contraire.

Quelques méthodes de ButtonGroup

- `void setSelected(int index)` – Sélectionne le RadioButton dont l'index est passé en paramètre.
- `void add(RadioButton rb)` – Ajoute un RadioButton au groupe.
- `void remove(RadioButton rb)` – Supprime un RadioButton du groupe.
- `int getButtonCount()` – Retourne le nombre de RadioButton à l'intérieur du ButtonGroup.
- `int getSelectedIndex()` – Retourne l'index du RadioButton sélectionné dans le groupe.
- `RadioButton getRadioButton(int index)` – Retourne le RadioButton dont l'index est passé en paramètre.
- `void clearSelection()` – Enlève les sélections. Cela revient à décocher tous les RadioButton à l'intérieur du groupe.

3.9. TextArea et TextField

TextArea

Ce composant permet de créer une zone de texte multiligne, éditable (état par défaut) ou non, pouvant contenir du texte sur plusieurs lignes. Par défaut, son contenu est alphanumérique, mais il est possible d'ajouter des contraintes pour n'accepter que certains types de contenus (par exemple : des chiffres, un e-mail, une adresse, etc.). Il existe au total sept constructeurs pour créer un TextArea. Voici quelques-uns d'entre eux :

Exemple : Zones de saisie avec ou sans contrainte

```
TextArea ta1=new TextArea(5, 10);
```

❶ `TextArea ta2=new TextArea("La vie est belle",5, 10);` **❷** `TextArea ta3=new TextArea(2, 10, TextArea.PASSWORD);` **❸** `TextArea ta4=new TextArea("Je suis super heureux", 30);` **❹**

❶ Création d'une zone de texte multiligne avec cinq lignes et dix colonnes.

❷ Création d'une zone de texte multiligne avec cinq lignes, dix colonnes et un texte par défaut à l'intérieur.

❸ Création d'une zone de texte multiligne avec deux lignes, dix colonnes et une contrainte. La contrainte `TextArea.PASSWORD` permet de préciser que ce champ acceptera un mot de passe. Cela aura pour effet de masquer les caractères entrés par l'utilisateur.

Création d'une zone de texte multiligne avec un texte par défaut et le nombre

- ❸ maximum de caractères acceptés dans ce champ. Ici, la zone de texte ne pourra donc pas accepter plus de trente caractères.

Figure 2.30 : Aperçu d'un TextArea avec et sans contrainte



Quelques méthodes de TextArea

- `void setText(String text)` – Ajoute du texte à la zone de texte ou modifie le texte existant de la zone de texte.
- `String getText()` – Retourne le texte à l'intérieur de la zone de texte.
- `String getTextAt(int ligne)` – Retourne le texte qui se trouve à la ligne dont la position est passée en paramètre.
- `void setEditable(boolean editable)` – Rend la zone de texte éditable ou non éditable. La valeur `true` la rend éditable et `false` non éditable.
- `void setGrowByContent(boolean grow)` – Si la valeur du paramètre est à `true`, cette méthode permettra d'agrandir automatiquement la hauteur de la zone de texte (au-delà du nombre de lignes spécifié) dans le but d'accueillir plus de texte.
- `void setRows(int rows)` – Indique le nombre de lignes de la zone de texte.

- `void setColumns(int columns)` – Indique le nombre de colonnes de la zone de texte.
- `void setRowsGap(int rowsGap)` – Indique l'espacement entre les lignes de la zone de texte. Cet espacement est exprimé en pixels.
- `void setMaxSize(int maxSize)` – Indique le nombre maximal de caractères à entrer dans la zone de texte.
- `int getLines()` – Retourne le nombre de lignes de texte dans la zone de texte.
- `void setHint(String hint)` – Permet d'ajouter un texte par défaut dans la zone de texte. Ce texte n'apparaît que lorsque la zone de texte est vide et disparaît dès que celle-ci est activée.
- `void setHintIcon(Image icon)` – Permet d'ajouter une image en arrière-plan dans la zone de texte. Cette image n'apparaît que lorsque la zone de texte est vide et disparaît dès que celle-ci est activée.
- `int getCursorPosition()` – Retourne la position du curseur à l'intérieur de la zone de texte.
- `void setConstraint(int constraint)` – Définit une contrainte. Cette contrainte permet de préciser le type de contenu que la zone de texte peut recevoir ou le comportement de ce contenu. Une contrainte est notée comme ceci : `TextArea.XXX` où `XXX` est la contrainte à appliquer. Les valeurs possibles du paramètre `constraint` sont récapitulées au [Tableau 2.1](#) de la [section Contraintes de saisie](#).

TextField

Tout comme `TextArea`, `TextField` permet de créer une zone de texte, mais d'une seule ligne uniquement. `TextField` hérite de `TextArea` et sur certaines plateformes il n'y a aucune différence entre les deux parce qu'il peut aussi accepter du texte sur plusieurs lignes. À la base, `TextField` a été inséré pour les besoins des plateformes J2ME, mais est fonctionnel sur les autres plateformes supportées par Codename One. À la différence d'un `TextArea`, un `TextField` contient un curseur clignotant.

Exemple : Zone de saisie d'une seule ligne

```
TextField nom=new TextField();
```

❶ `nom.setHint("Entrez votre nom");` **❷** `TextField identifiant=new TextField("Entrez votre identifiant");` **❸** `TextField motdepasse=new TextField();`
`motdepasse.setConstraint(TextField.PASSWORD);` **❹**

❶ Création du `TextField`.

② La méthode `setHint()` permet d'ajouter le texte “Entrez votre nom” en arrière-plan du `TextField`. Ce texte disparaîtra automatiquement une fois la zone de texte activée.

③ Création d'un `TextField` avec insertion d'un texte à l'intérieur de la zone de texte.

La méthode `setConstraint()` permet de préciser le type de contenu que la zone de
④ texte peut recevoir. En l'occurrence, il s'agit d'un mot de passe, donc le contenu sera masqué.

Figure 2.31 : Aperçu d'un `TextField`



Quelques méthodes de `TextField`

- `void setText(String text)` – Insère du texte ou modifie le texte à l'intérieur du `TextField`.
- `String getText()` – Retourne le contenu ou texte du `TextField`.
- `void clear()` – Permet de supprimer le contenu du `TextField`.
- `int getCursorPosition()` – Retourne la position du curseur dans le `TextField`.
- `void setCursorPosition(int pos)` – Permet de définir la position du curseur.
- `void setEditable(boolean editable)` – Rend la zone de texte éditable ou non éditable. La valeur `true` la rend éditable et `false` non éditable.
- `void setInputMode(String inputMode)` – Permet de définir le mode d'entrée par défaut du `textField`. Les valeurs possibles du paramètre `inputMode` sont : `Abc`, `ABC`, `abc`, `123`.
- `void addDataChangedListener(DataChangedListener listener)` – Permet de s'abonner à un `DataChangedListener` pour détecter l'entrée de données (ou de caractères) en temps réel dans le `TextField`.

Contraintes de saisie

Des contraintes de saisie peuvent être appliquées à TextArea et à TextField avec la méthode `setConstraint()`. Le [Tableau 2.1](#) liste les contraintes disponibles.

Tableau 2.1 : Contraintes applicables à une zone de texte

ANY	Accepte tous les types de données. C'est la valeur par défaut.
DECIMAL	Contrainte adaptée uniquement aux nombres avec la possibilité d'insérer des nombres à virgule.
EMAILADDR	Contrainte adaptée aux e-mails.
INITIAL_CAPS_SENTENCE	Indique que le premier caractère de chaque phrase doit être en majuscule.
INITIAL_CAPS_WORD	Indique que le premier caractère de chaque mot doit être en majuscule.
NON_PREDICTIVE	Indique à la zone de texte de ne pas utiliser le dictionnaire de suggestion de mots pendant la saisie.
NUMERIC	Contrainte adaptée aux nombres.
PASSWORD	Contrainte adaptée aux mots de passe. L'effet de cette contrainte est de masquer les caractères entrés dans la zone de texte.
PHONENUMBER	Contrainte adaptée aux numéros de téléphone.
SENSITIVE	Indique que le contenu de la zone de texte est sensible et ne doit pas être stocké dans le dictionnaire de suggestion de mots.
UNEDITABLE	Indique que le contenu de la zone de texte est non éditable.
URL	Contrainte adaptée aux adresses.

3.10. AutoCompleteTextField

Un AutoCompleteTextField est semblable à un TextField avec en plus la possibilité de proposer à l'utilisateur pendant sa saisie une liste de suggestions de mots ou de phrases. En bref, ce composant propose de l'autocomplétion. Le fait qu'il hérite de TextField lui donne automatiquement accès aux méthodes de cette classe.

Exemple : Zone de saisie avec autocomplétion

```
String[] listeVilles=new String[]{"Lomé", "Paris", "Londres", "Santo Domingo", "Bangkok", "Pattaya", "Punta Cana"};
```

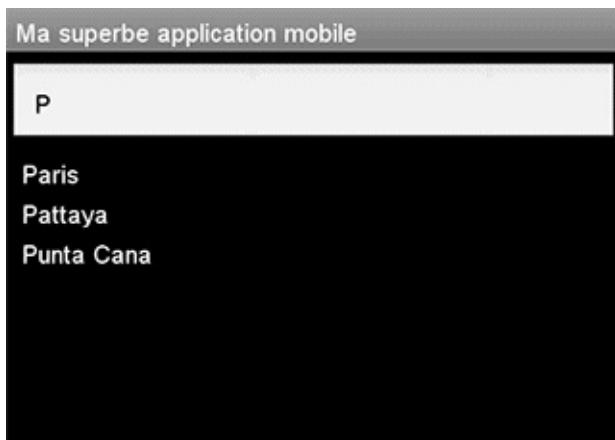
① AutoCompleteTextField villes=new AutoCompleteTextField(listeVilles); ②

① Création d'un tableau de chaînes de caractères avec des noms de villes.

② Création d'un AutoCompleteTextField avec en paramètre le tableau de chaînes de caractères créé précédemment.

Le constructeur du composant AutoCompleteTextField peut aussi prendre en paramètre un objet de type ListModel<String>.

Figure 2.32 : Aperçu d'un AutoCompleteTextField



Quelques méthodes d'AutoCompleteTextField

- void setCompletionRenderer(ListCellRenderer completionRenderer) – Permet de définir un rendu personnalisé pour la liste de suggestion des mots. Nous verrons plus de détails sur la classe ListCellRenderer du paramètre de cette méthode avec le composant List.

3.11. ComboBox

Le ComboBox permet de créer une liste déroulante pouvant contenir des données alphanumériques. Ce composant est une combinaison d'une liste et d'un bouton où il n'est possible de choisir qu'un élément à la fois.

Exemple : Liste déroulante

```
ComboBox cb1=new ComboBox();
```

❶ ❷ cb1.addItem("Eric"); cb1.addItem("Patricia"); ComboBox cb2=new ComboBox(new String[]{ "CodenameOne", "Titanium", "Xamarin" }); ❸

❶ Création d'un ComboBox vide.

❷ Ajout des éléments dans le ComboBox avec la méthode `addItem()`.

❸ Création d'un ComboBox avec en paramètre un tableau de chaînes de caractères.

Le constructeur de ComboBox peut aussi prendre en paramètre un tableau de type Vector ou l'une des classes enfants de ListModel.

Figure 2.33 : Sous Android 4.x

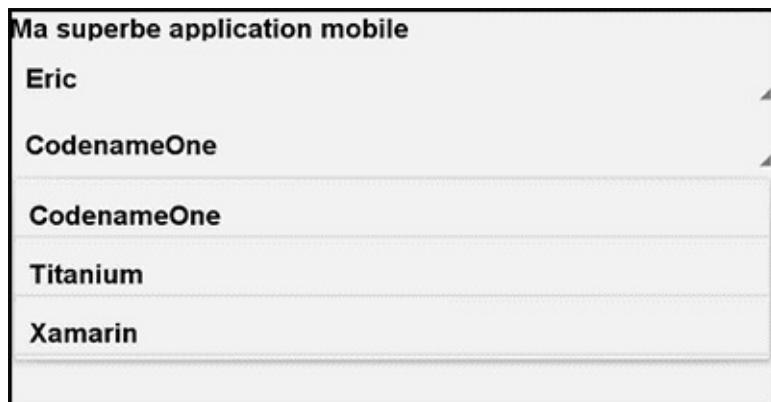
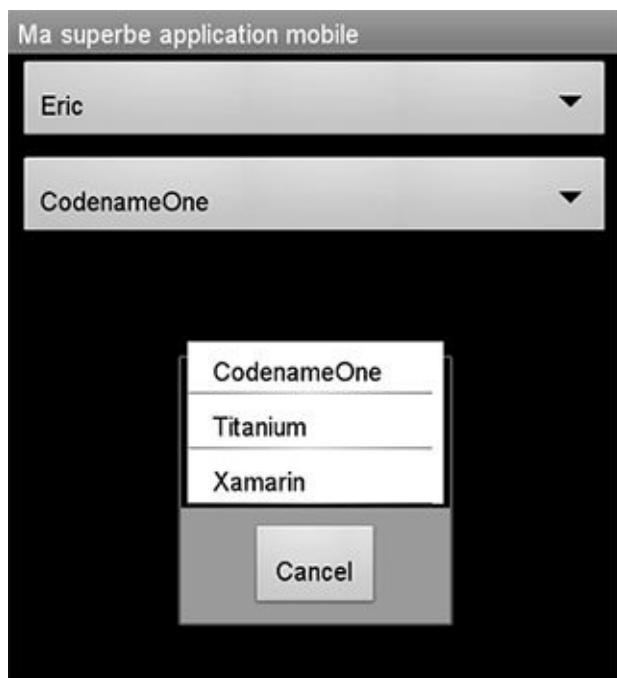


Figure 2.34 : Sous Android 2.x



Quelques méthodes de ComboBox

- `void setSelectedIndex(int index)` – Permet de définir l’élément courant de la liste déroulante. Il faut passer en paramètre à cette méthode la position de l’élément à choisir.
- `int getSelectedIndex()` – Retourne la position de l’élément sélectionné dans la liste déroulante.
- `Object getSelectedItem()` – Retourne l’élément sélectionné dans la liste déroulante.

3.12. Slider

Si vous avez besoin d’un composant pour indiquer la progression d’une opération en cours ou tout simplement pour permettre à l’utilisateur de sélectionner une valeur parmi des valeurs prédéfinies dans une liste, alors le composant Slider vous rendra de fiers services. Un Slider peut être horizontal (position par défaut) ou vertical et sa valeur peut être affichée sous forme de pourcentage ou de chiffre. Il peut être édité ou non par l’utilisateur.

Exemple : Jauge interactive

Le code suivant crée une interface avec deux boutons et une jauge. L’un des boutons permettra d’augmenter la valeur de la jauge et l’autre de la diminuer.

```
Form f = new Form("Ma superbe application mobile");
BoxLayout layout= new BoxLayout(BoxLayout.Y_AXIS);
f.setLayout(layout);

final Slider s=new Slider();

❶ s.setRenderPercentageOnTop(true); ❷ Container c=new Container(new
FlowLayout(Component.CENTER)); Button b=new Button("Augmenter"); ❸
b.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent
evt) { s.setProgress(s.getProgress()+1); ❹ } }); Button b2=new Button("Diminuer"); ❺
b2.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent
evt) { s.setProgress(s.getProgress()-1); ❻ } }); c.addComponent(b); c.addComponent(b2);
f.addComponent(c); f.addComponent(s); f.show();
```

❶ Création de la jauge avec un Slider.

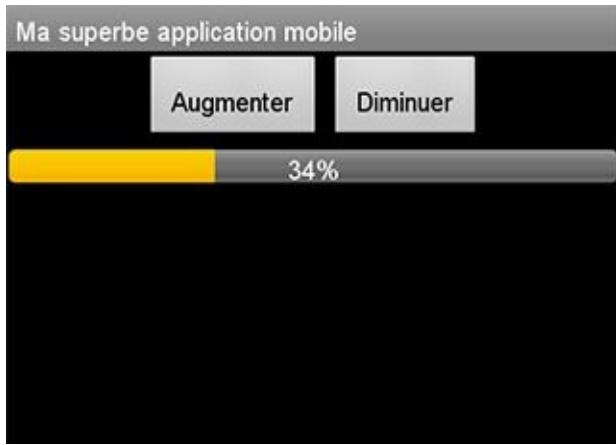
❷ La méthode `setRenderPercentageOnTop()` passée à true indique au Slider d’afficher sa valeur courante en pourcentage au cours de sa progression.

❸ Création du premier bouton qui servira à augmenter la valeur de progression du Slider.

La méthode `setProgress()` permet de définir la valeur courante du Slider et

- ❸ `getProgress()` de la récupérer. Dans notre code, la nouvelle valeur du Slider est égale à son ancienne valeur +1.
- ❹ Création du deuxième bouton qui servira à diminuer la valeur de progression du Slider.
- ❺ À l'opposé du point ❸, la nouvelle valeur du Slider ici est égale à son ancienne valeur -1.

Figure 2.35 : Aperçu d'une jauge interactive



Quelques méthodes de Slider

- `void setMinValue(int minValue)` – Permet d'indiquer la valeur minimum du Slider.
- `void setMaxValue(int maxValue)` – Permet d'indiquer la valeur maximum du Slider.
- `void setProgress(int value)` – Permet d'indiquer la valeur de progression.
- `void setIncrements(int increments)` – Incrémente la valeur du Slider lorsqu'on appuie sur les touches gauche/droite/haut/bas du téléphone.
- `void setEditable(boolean editable)` – Indique si le Slider est éditable manuellement ou non par l'utilisateur. Si oui, l'utilisateur peut changer lui-même sa valeur.
- `void setInfinite(boolean infinite)` – Si la valeur du paramètre `infinite` de cette méthode est à `true` alors le Slider affichera une animation de progression infinie sans afficher de valeur précise.
- `void setVertical(boolean b)` – Indique si le Slider sera vertical ou non. La position par défaut est horizontale.
- `void setRenderPercentageOnTop(boolean b)` – Indique si la valeur en pourcentage du Slider doit être affichée ou non.

- `void setRenderValueOnTop(boolean renderValue)` – Indique si la valeur du Slider doit être affichée ou non.
- `int getProgress()` – Retourne la valeur de progression actuelle du Slider.
- `void addDataChangedListener(DataChangedListener listener)` – Permet de s'abonner à un DataChangedListener pour détecter les changements de valeurs du Slider.

3.13. Calendar

Le composant Calendar fournit un calendrier visuel permettant à l'utilisateur de choisir une date. Ce composant hérite de la classe Container et affiche par défaut les jour, mois et année courante.

Exemples : Calendriers avec différentes options

```
Calendar cal=new Calendar();
```

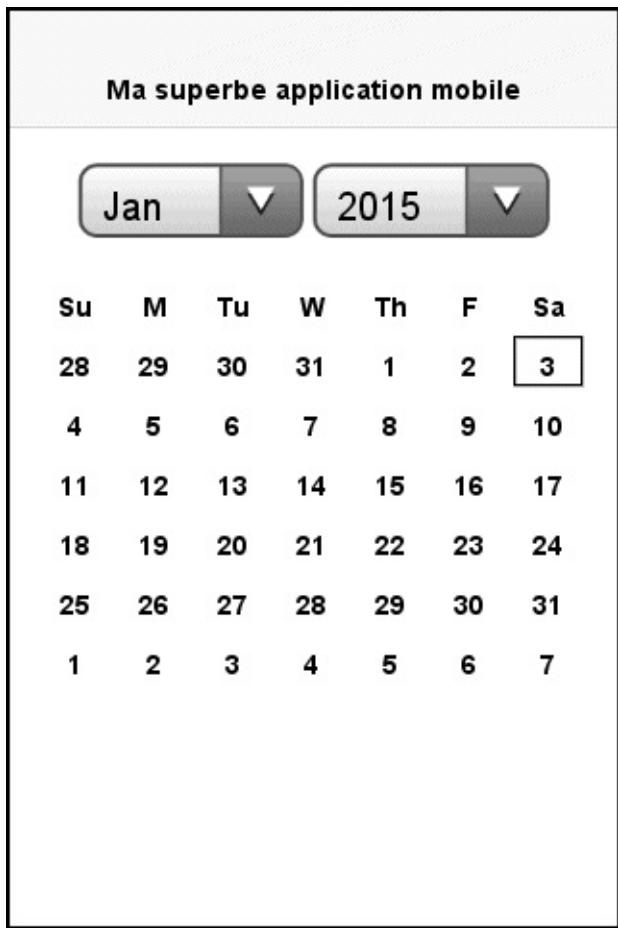
❶ `Calendar cal2=new Calendar(10000);` **❷** `Calendar cal3=new Calendar(10000,TimeZone.getDefault());` **❸**

❶ Création d'un calendrier avec la date courante par défaut.

Création d'un calendrier basé sur le temps écoulé depuis le 1^{er} janvier 1970 à **❷** 00h00 GMT (calendrier grégorien). Le temps passé en paramètre est exprimé en millisecondes.

Création d'un calendrier basé sur le temps écoulé depuis le 1^{er} janvier 1970 à **❸** 00h00 GMT et sur un fuseau horaire défini. Le fuseau horaire est défini en second paramètre et est géré par la classe TimeZone.

Figure 2.36 : Aperçu d'un calendrier avec la date du jour



Quelques méthodes de Calendar

- `Date getCurrentDate()` – Retourne la date courante du calendrier.
- `Date getDate()` – Retourne la date sélectionnée par l'utilisateur.
- `TimeZone getTimeZone()` – Retourne le fuseau horaire.
- `void setDate(Date)` – Définit la date courante que le calendrier affichera par défaut.
- `void setYearRange(int minYear, int maxYear)` – Définit l'intervalle de dates que doit contenir le calendrier. Le paramètre `minYear` définit l'année minimum et `maxYear` l'année maximum.
- `void setChangesSelectedDateEnabled(boolean enabled)` – Permet d'activer ou de désactiver le fait de pouvoir changer ou non la date du calendrier par l'utilisateur.
- `void addDataChangedListener(DataChangedListener listener)` – Permet de s'abonner à un `DataChangedListener` pour détecter les changements de date, mois et année dans le calendrier.

3.14. WebBrowser

Ce composant permet d'afficher un navigateur web capable d'interpréter et d'afficher une page web HTML. WebBrowser encapsule en fait deux autres composants qui sont BrowserComponent et HTMLComponent. Le premier permet d'utiliser le composant natif de navigation web sur les plateformes qui en ont un (iOS, Android et BlackBerry par exemple) et le second permet d'utiliser le navigateur web présent dans Codename One sur les plateformes qui n'en ont pas en natif. Ainsi, en fonction de la plateforme sur laquelle s'exécutera votre application, WebBrowser sélectionnera automatiquement le composant approprié à utiliser.

Exemple : Lire une page HTML locale ou distante

```
WebBrowser wb=new WebBrowser();
```

❶ wb.setURL("jar:///mapage.html"); ❷ WebBrowser wb=new WebBrowser();
wb.setURL("http://mobile.twitter.com"); ❸

❶ Création d'une fenêtre de navigation avec WebBrowser.

❷ Chargement d'un fichier HTML (mapage.html) placé dans le dossier SRC du projet. À la compilation, ce fichier HTML sera intégré à l'exécutable.

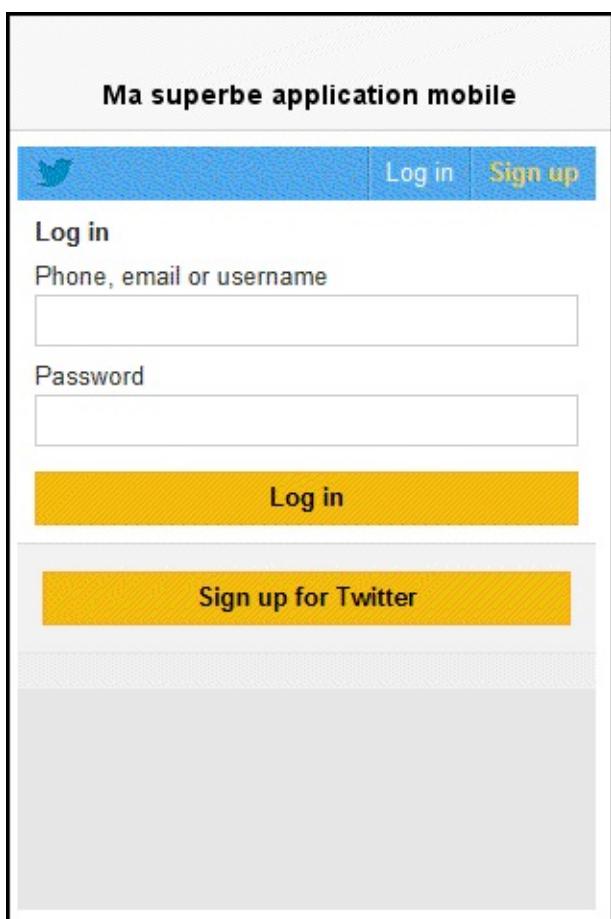
❸ Chargement de la version mobile du site de Twitter.

La méthode setURL() permet de charger un fichier local ou un site en ligne.

Figure 2.37 : Lecture d'une page HTML locale avec WebBrowser



Figure 2.38 : Lecture d'une page web distante avec WebBrowser



Quelques méthodes de WebBrowser

- void reload() – Permet d'actualiser la page courante ouverte dans le WebBrowser.
- void stop() – Permet de stopper le chargement d'une page.
- String getURL() – Retourne l'adresse de la page web chargée.
- String getTitle() – Retourne le titre de la page web chargée dans le WebBrowser.
- void onLoad(String url) – Méthode callback qui est appelée à chaque fois que le chargement de l'adresse spécifiée par setURL(String url) est terminé. Vous pouvez implémenter cette méthode pour exécuter une action quelconque.
- void onStart(String url) – Méthode callback qui est appelée à chaque fois avant le chargement de l'adresse à charger. Vous pouvez implémenter cette méthode pour exécuter une action quelconque.
- void onError(String message, int errorCode) – Méthode callback qui est appelée à chaque fois qu'une erreur de chargement survient. Vous pouvez implémenter cette méthode pour exécuter une action quelconque.

3.15. Picker

Picker permet de créer un sélecteur de données, un peu dans la logique d'une liste déroulante. Un Picker peut être utilisé pour afficher une liste de dates, une liste d'heures, une liste de dates et d'heures combinées ou tout simplement une liste de mots ou encore de chiffres. Avant l'arrivée de ce composant dans Codename One, il fallait utiliser une classe différente pour créer chacun de ces sélecteurs. Avec Picker, il suffit de préciser le type de sélecteur qu'on veut, et ce sera fait. Visuellement, Picker se présente comme une zone de texte non éditable qui affiche une petite fenêtre pop-up contenant les éléments à choisir quand on clique dessus. Si Picker est disponible en natif sur la plateforme sur laquelle s'exécute l'application, cet équivalent natif sera utilisé. Dans le cas contraire, le sélecteur sera dessiné par Codename One et aura la même apparence sur les plateformes qui ne l'ont pas en natif dans leur système.

Exemple : Sélecteurs de date et/ou heure

```
Picker p=new Picker();
```

```
❶ p.setType(Display.PICKER_TYPE_DATE); ❷ Picker p2=new Picker();
p2.setType(Display.PICKER_TYPE_TIME); ❸ Picker p3=new Picker();
p3.setType(Display.PICKER_TYPE_DATE_AND_TIME); ❹ Picker p4=new Picker();
p4.setType(Display.PICKER_TYPE_STRINGS); ❺ p4.setStrings(new String[]
{“CodenameOne”, “Titanium”, “Xamarin”}); ❻
```

❶ Création du Picker.

❷ Définition du type de Picker voulu. Sur cette ligne, nous choisissons un Picker qui affichera des dates.

❸ Ici, nous précisons que ce Picker affichera l'heure.

❹ Ici, nous précisons que ce Picker affichera la date et l'heure.

❺ Ici, nous précisons que ce Picker affichera du texte comme le ferait un ComboBox par exemple.

❻ Ajout de trois mots à l'intérieur du Picker avec la méthode `setStrings()`.

Figure 2.39 : Aperçu de différents Pickers

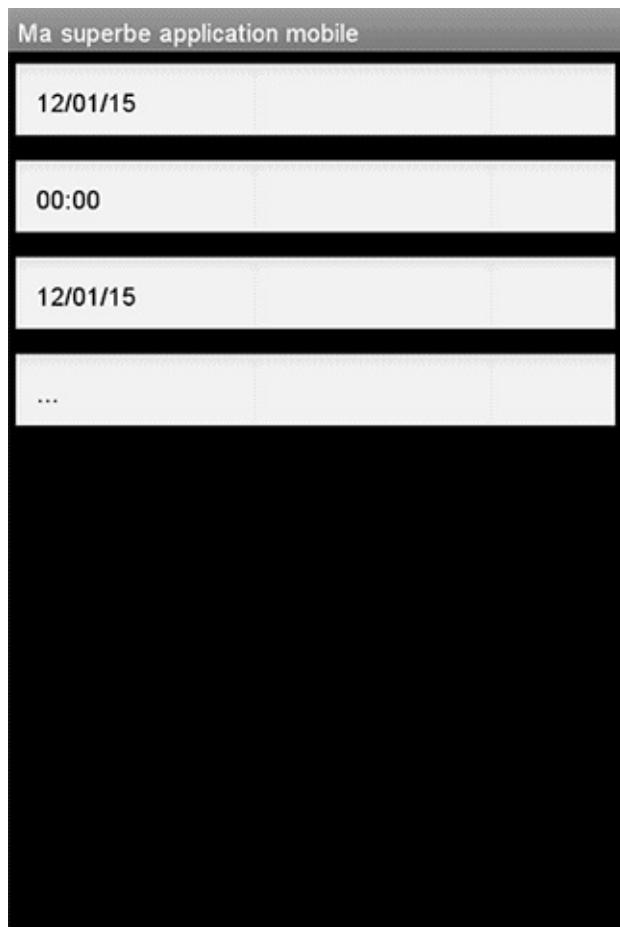
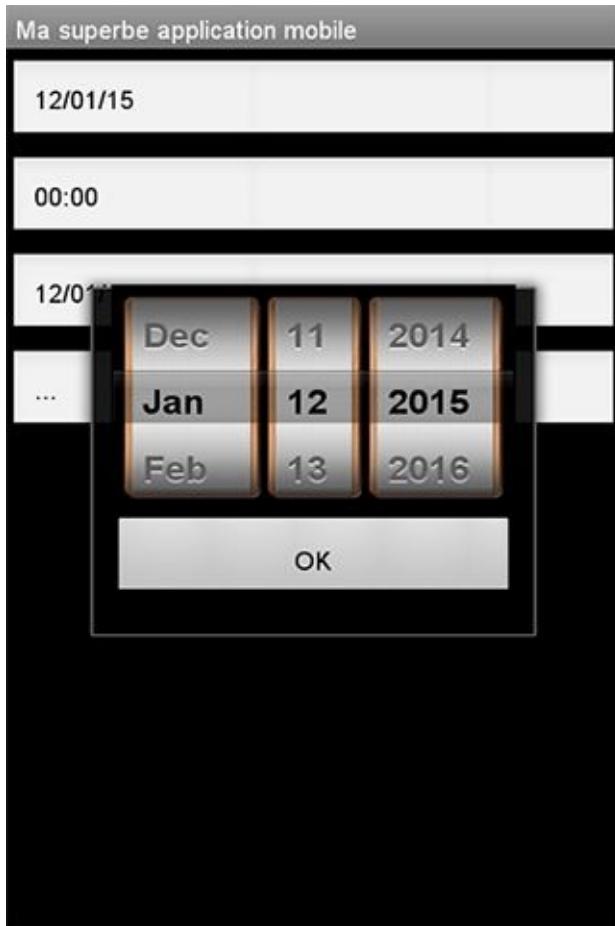


Figure 2.40 : Affichage des données d'un Picker



Quelques méthodes de Picker

- `void setType(int type)` – Permet de définir le type de Picker qu'on veut créer. Le paramètre `type` peut prendre l'une des quatre valeurs suivantes :
`Display.PICKER_TYPE_DATE`, `Display.PICKER_TYPE_TIME`,
`Display.PICKER_TYPE_DATE_TIME`, `Display.PICKER_TYPE_STRINGS`.
- `int getType()` – Retourne le type du Picker.
- `void setDate(Date date)` – Permet de définir la date par défaut du Picker. Ceci est valable pour les types `Display.PICKER_TYPE_DATE` et
`Display.PICKER_TYPE_DATE_TIME` uniquement.
- `Date getDate()` – Retourne la date sélectionnée dans le Picker. Uniquement pour les types `Display.PICKER_TYPE_DATE` et `Display.PICKER_TYPE_DATE_TIME`.
- `void setTime(int time)` – Permet de définir l'heure pour un Picker de type `Display.PICKER_TYPE_TIME` uniquement. Le paramètre `time` est le nombre de minutes écoulées depuis 00h00.
- `int getTime()` – Retourne l'heure sélectionnée dans le Picker de type `Display.PICKER_TYPE_TIME` uniquement.
- `void setShowMeridiem(boolean showMeridiem)` – Permet d'indiquer si l'heure doit

être affichée au format AM/PM (paramètre `showMeridiem` à `true`) ou sous le format de 24 heures (paramètre `showMeridiem` à `false`) qui est le format par défaut.

3.16. OnOffSwitch

Ce composant peut être comparé à un CheckBox, mais avec un style particulier. Il ne peut avoir que deux états (On et Off). Son apparence est totalement différente sous iOS et Android parce qu'il s'agit d'un composant natif sous ces deux plateformes. Il fonctionne aussi sur les autres plateformes (BlackBerry et Windows Phone), qui ne le proposent pas nativement. L'état par défaut de ce composant est On.

Exemple : Bouton Marche/Arrêt

```
OnOffSwitch oos=new OnOffSwitch();
```

❶ `oos.setValue(true);` ❷

❶ Création d'un bouton OnOffSwitch.

❷ La méthode `setValue()` qui prend un booléen en paramètre permet d'activer ou de désactiver par défaut le OnOffSwitch. Quand elle est activée, sa valeur est sur On et quand elle est désactivée, sa valeur est sur Off.

Figure 2.41 : Un bouton Marche/Arrêt (sous iOS)



Quelques méthodes de OnOffSwitch

- `void setOn(String onText)` – Change le texte qui sera affiché quand le bouton sera sur l'état On. Par défaut, le texte affiché est On sur certaines plateformes. Sur d'autres, il n'y a aucun texte qui s'affiche mais juste un visuel qui désigne les deux états que peut prendre ce bouton.
- `void setOff(String offText)` – Change le texte qui sera affiché quand le bouton sera sur l'état Off. Par défaut, le texte affiché est Off sur certaines plateformes. Sur

d'autres, il n'y a aucun texte qui s'affiche mais juste un visuel qui désigne les deux états que peut prendre ce bouton.

- `boolean isValue()` – Retourne la valeur courante du OnOffswitch.
- `String getOn()` – Retourne le texte défini pour l'état On du bouton.
- `String getOff()` – Retourne le texte défini pour l'état Off du bouton.

3.17. ShareButton

Un ShareButton est un bouton spécial qui permet de partager des informations avec les amis et proches depuis une application. Ce partage peut être fait par les réseaux sociaux ou par les canaux habituels comme le SMS, l'e-mail et autres. Il est possible de personnaliser ce bouton et d'ajouter d'autres services de partage.

Note > Sous iOS et Android, ce bouton affichera les différentes options de partage disponibles en natif sur ces plateformes. Sur les autres plateformes, les choix de partage qui s'afficheront sont Facebook, SMS et l'e-mail.

Exemple : Bouton de partage de données

```
ShareButton sb=new ShareButton();
```

❶ `sb.setText("Invitez vos amis");` ❷ `sb.setTextToShare("Bonjour, je t'invite à télécharger et à installer l'application nommée XXX. C'est un truc de ouf pour...");` ❸

❶ Création du ShareButton.

❷ La méthode `setText()` permet de définir le texte du bouton.

❸ La méthode `setTextToShare()` permet de définir le texte qui sera partagé et envoyé à d'autres personnes.

Figure 2.42 : Avant le clic sur le ShareButton

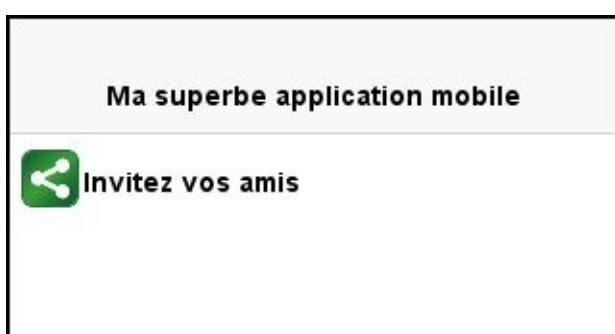


Figure 2.43 : Après le clic sur le ShareButton (sous une plateforme différente de iOS et Android)



Quelques méthodes de ShareButton

- `void setImageToShare(String imagePath, String imageMimeType)` – Permet de définir l'image à partager. Le premier paramètre `imagePath` est le chemin complet vers l'image à partager et le second paramètre `imageMimeType` définit le type MIME de l'image.
- `String getTextToShare()` – Renvoie le texte à partager.
- `String getImagePathToShare()` – Renvoie le chemin vers l'image à partager.
- `void addShareService(ShareService share)` – Permet d'ajouter d'autres services de partage en sus des trois proposés par défaut (Facebook, SMS et Email). Il est possible d'ajouter son propre service de partage en créant une classe qui va hériter de la classe `ShareService` et en implémentant les fonctionnalités voulues. Pour vous exercer, vous pouvez par exemple créer un service pour partager des images sur Flickr ou des messages sur Twitter. Nous n'aborderons pas dans ce livre la création d'un `ShareService` mais après la lecture du chapitre [Réseau, Internet et services web](#), vous serez en mesure de créer ce genre de choses.

3.18. ImageViewer

ImageViewer est un composant qui permet de créer un effet de slide avec des images. Chacune des images peut être zoomée.

Exemple : Créer un carrousel d'images

```
ImageViewer iv=new ImageViewer();
```

```
❶ DefaultListModel<Image> listeImages=new DefaultListModel<Image>(); ❷ try{ ❸  
listeImages.addItem(Image.createImage("/slide1.jpg"));  
listeImages.addItem(Image.createImage("/slide2.jpg")); } catch(IOException ex){  
Dialog.show("Erreur", ex.getMessage(), "OK",null); }  
iv.setImage(listeImages.getItemAt(0)); ❹ iv.setImageList(listeImages); ❺
```

❶ Création du ImageViewer.

❷ Création d'une liste d'images qui va contenir les images à insérer dans le diaporama.
Liste créée avec la classe DefaultListModel.

❸ Ajout de deux images dans la liste d'images.

❹ Définition de l'image de la liste qui sera affichée en premier avec la méthode
setImage().

❺ Ajout de la liste d'images au diaporama avec la méthode setImageList().

Figure 2.44 : Aperçu de l'effet de slide

Ma superbe application mobile



Ma superbe application mobile



Quelques méthodes de ImageViewer

- `void setSwipePlaceholder(Image placeholderImg)` – Définit l'image qui sera affichée en attendant que les images du slide soient chargées. Cette méthode est utile en cas de lenteur du chargement des images.
- `void setZoom(float zoom)` – Permet de définir le niveau de zoom.
- `void setCycleLeft(boolean cycleleft)` – Permet d'activer ou de désactiver le fait de défiler les images du slide en partant de la gauche.
- `void setCycleRight(boolean cycleright)` – Permet d'activer ou de désactiver le fait de défiler les images du slide en partant de la droite.
- `Image getImage()` – Retourne l'image affichée par le slide.
- `ListModel<Image> getImageList()` – Retourne la liste des images contenues dans le slide.

3.19. MapComponent

Nous allons maintenant toucher à la géolocalisation avec ce composant. MapComponent permet d'afficher une carte sur laquelle on peut naviguer et localiser des endroits. Il affiche une carte basée sur OpenStreetMap. Si vous voulez afficher une carte basée sur Google Maps alors vous allez devoir faire appel à un plug-in CodenameOne parce que le support de Google Maps n'est pas intégré à l'API interne. Pour plus d'informations sur les plug-ins de Codename One, allez à la section [Le système de plug-in de Codename One \(CN1LIB\)](#) du chapitre [Plug-ins et code natif](#).

Exemple : Localiser l'utilisateur sur OpenStreetMap

Dans l'exemple suivant, nous allons localiser l'utilisateur de notre application de test sur la carte d'OpenStreetMap à partir de ses coordonnées, lesquelles sont représentées par la longitude et la latitude de son emplacement.

```
MapComponent map=new MapComponent();  
  
❶ try { Image img=Image.createImage("/location.png"); ❷ Location  
location=LocationManager.getLocationManager().getCurrentLocationSync(); ❸ Coord  
coordonnees=new Coord(location.getLatitude(),location.getLongitude()); ❹ PointLayer  
pl=new PointLayer(coordonnees, "Hello! Vous êtes ici", img); ❺  
pl.setDisplayName(true); ❻ PointsLayer pointsL=new PointsLayer(); ❻  
pointsL.addPoint(pl); ❽ pointsL.setPointIcon(img); ❾ map.addLayer(pointsL); ❿ } catch  
(IOException ex) { Dialog.show("Erreur", ex.getMessage(), "Ok", null); }  
map.zoomToLayers(); ❾
```

Comme vous pouvez le remarquer dans cet exemple, MapComponent fonctionne en association avec d'autres classes que je vais vous présenter. Sur la première ligne du code **❶**, nous créons le MapComponent qui gère l'affichage de la carte. Sitôt après, une image est chargée **❷**. Nous l'utiliserons plus tard pour marquer sur la carte où se trouve l'utilisateur.

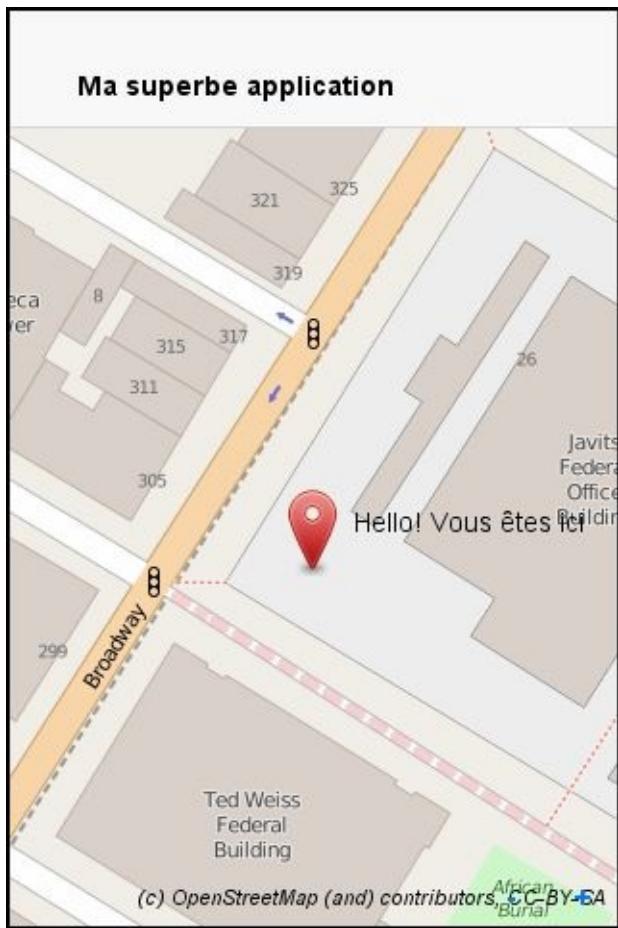
Nous faisons ensuite appel à la classe LocationManager pour récupérer sa position (ou la position de son téléphone si vous préférez) avec la méthode `getCurrentLocationSync()` **❸**. Celle-ci retourne un objet de type Location qui gère la localisation même. `getCurrentLocationSync()` est bloquante et ne retourne qu'à la fin de son exécution. Son équivalent non bloquant (asynchrone) est `getCurrentLocation()`.

Nous créons ensuite une variable de type Coord qui va prendre en paramètres dans son constructeur la longitude et la latitude de sa position **❹**, qui sont fournies par la variable de type Location retournée par `getCurrentLocationSync()`.

En **❺**, nous créons un calque avec la classe PointLayer. Les calques permettent d'afficher des images, des formes ou encore du texte sur un emplacement de la carte. Ainsi, dans cet exemple, notre calque contient les coordonnées, un texte à afficher et une image à placer à l'endroit où se trouve l'utilisateur. La méthode `setDisplayName()`, dont le paramètre est passé à `true`, permet de confirmer l'affichage de sa position sur la carte **❻**. Il n'est pas possible d'ajouter le calque PointLayer directement à notre carte. Pour cela, nous devons d'abord créer une liste de calques avec la classe PointsLayer **❼** puis ajouter notre calque à cette liste **❽**, qui sera elle-même ajoutée ensuite à la carte **❾** avec la méthode `addLayer()` de la classe MapComponent. La méthode `setPointIcon()` de PointsLayer permet de définir l'image à placer à la position de l'utilisateur **❾**.

Pour finir, la méthode `zoomToLayers()` de MapComponent permet de zoomer sur la zone contenant les calques **❿**.

Figure 2.45 : MapComponent affichant la localisation de l'utilisateur



Quelques méthodes usuelles de MapComponent

- `void addLayer(Layer layer)` – Ajoute un calque à la carte.
- `void addLayer(Layer layer, int minzoomlevel, int maxZoomLevel)` – Ajoute un calque à la carte tout en précisant le niveau de zoom minimum et le niveau de zoom maximum de la carte.
- `int getLayersCount()` – Retourne le nombre total de calques sur la carte.
- `Layer getLayerAt(int index)` – Retourne le calque dont l'index est passé en paramètre.
- `Coord center()` – Retourne les coordonnées (longitude et latitude) du centre de la carte.
- `void zoomIn()` – Permet de zoomer la carte.
- `void zoomOut()` – Permet de dézoomer la carte.
- `void zoomTo(Coord coord, int zoomLevel)` – Permet de zoomer la carte sur les coordonnées passées en premier paramètre. Le second paramètre permet de définir le niveau de zoom.
- `void setZoomLevel(int zoom)` – Permet de définir le niveau de zoom de la carte.

- void moveUp() – Permet de déplacer de 25 % la carte vers le haut.
- void moveDown() – Permet de déplacer de 25 % la carte vers le bas.
- void moveRight() – Permet de déplacer de 25 % la carte vers la droite.
- void moveLeft() – Permet de déplacer de 25 % la carte vers la gauche.

3.20. List

Le composant List permet de présenter une liste d'éléments dans une colonne. Les éléments peuvent être présentés verticalement ou horizontalement. Par défaut, les éléments d'une List sont des textes bruts, mais en raison de sa flexibilité ils peuvent aussi provenir de n'importe quel autre composant (Label, TextArea, etc.). En bref, le rendu peut être personnalisé à volonté. Cela est possible parce que ce composant est basé sous l'architecture MVC (Modèle Vue Contrôleur) qui permet de séparer la vue et les données.

- Le modèle représente les données qui seront ajoutées à la liste et fournit des informations comme le nombre d'éléments à ajouter par exemple. Ce modèle peut être un tableau de Vector, un tableau de chaînes de caractères ou un objet qui hérite de l'interface ListModel. Le modèle le plus basique est DefaultListModel.
- La vue permet d'afficher le contenu du modèle. Cette action est effectuée chaque fois qu'un changement est effectué dans le modèle. Le rendu est géré par l'interface ListCellRenderer.
- Le contrôleur s'occupe de récupérer des données entrées par l'utilisateur pour modifier le contenu du modèle. Cette modification affectera automatiquement la vue qui déclenchera une mise à jour du rendu. Le composant List en soi constitue le contrôleur.

Création d'une liste simple

Dans l'exemple suivant, nous allons créer une liste de mots avec le modèle DefaultListModel.

Exemple 2.3 : Création d'une liste de mots

```
Form f=new Form("Hi World");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
f.setScrollable(false);
```

```
❶ DefaultListModel model=new DefaultListModel();
❷ ❸ model.addItem("Thaïlande");
model.addItem("Japon"); model.addItem("République dominicaine");
model.addItem("Nouvelle Zélande"); List listePays=new List(model);
❹ listePays.setFixedSelection(List.FIXED_NONE_CYCLIC); ❺
f.addComponent(listePays); f.show();
```

Ici, nous désactivons le défilement du Form pour une raison bien précise. Par défaut, les composants List et Form activent automatiquement le défilement dès qu'il y a un dépassement de données au niveau des dimensions de l'écran. Le fait d'avoir une liste

❶ qui défile à l'intérieur d'un Form qui a lui-même le défilement activé cause un comportement bizarre à l'écran et empêche d'atteindre correctement les données qui sont en dessous de la liste. Pour cela, il faut toujours désactiver le défilement du Form pour qu'il n'entre pas en conflit avec celui de List.

❷ Création d'un objet de DefaultListModel qui va contenir les données à afficher. Ici, il s'agit d'une liste de pays.

Ajout des noms de pays à l'objet DefaultListModel avec la méthode addItem(). On ❸ aurait pu faire la même chose en créant un Vector ou un simple tableau de chaînes de caractères et en passant ça en paramètre au constructeur de DefaultListModel.

Nous créons l'objet List et nous lui passons en paramètre l'objet DefaultListModel pour insérer dans la liste les données à l'intérieur du modèle. La même chose peut se faire avec la méthode setModel() de la classe List qui prend l'objet du modèle en paramètre.

Par défaut, lorsqu'on parcourt les éléments d'une liste avec une sélection et qu'on arrive par exemple au dernier élément de la liste, la sélection s'arrête et ne reprend pas automatiquement à partir du premier élément. C'est pour obtenir ce comportement que ❹ nous utilisons ici la méthode setFixedSelection() qui prend en paramètre la valeur FIXED_NONE_CYCLIC pour rendre la sélection cyclique. D'autres valeurs intéressantes sont disponibles.

Figure 2.46 : Rendu de List sous iOS 8



Figure 2.47 : Rendu de List sous Android 4.x

Ma superbe application mobile

Thaïlande

Japon

République dominicaine

Nouvelle zélande

Au lieu de passer par un modèle, il est aussi possible d'ajouter directement la liste des pays au composant List via sa méthode `addItem()`. Cette manière de faire est cependant limitée car le composant List ne contient pas de méthodes pour supprimer un ou plusieurs éléments de la liste, pour ajouter un élément à position spécifique, pour connaître la taille du nombre d'éléments, etc. Ainsi, ce n'est qu'en passant par un modèle qu'il est possible d'effectuer ces actions qui y sont implémentées.

Création d'une liste complexe (Personnalisation du rendu)

Comme mentionné dans le paragraphe d'introduction, il est possible de remplacer les textes d'une liste par des composants plus élaborés ou tout simplement de personnaliser le rendu de ces textes. Pour nous aider, nous ferons appel à l'interface ListCellRenderer. Cette interface dispose de deux méthodes à implémenter qui sont les suivantes :

- Component `getListCellRendererComponent(List list, Object value, int index, boolean isSelected)` – Cette méthode est invoquée sur chaque élément de la liste quand aucune action n'est effectuée c'est-à-dire à l'état visuel par défaut. Son quatrième paramètre `isSelected` renvoie `true` chaque fois qu'un élément de la liste est sélectionné. Il faut se servir de sa valeur pour définir des rendus différents quand un élément est sélectionné ou non. Le premier paramètre représente le composant List en soi, le deuxième paramètre est la valeur d'un élément de la liste et le troisième `index` est l'indice d'un élément de la liste.
- Component `getListFocusComponent(List list)` – Cette méthode est invoquée sur chaque élément de la liste chaque fois qu'une sélection passe dessus (différente de l'état sélectionné du composant).

Dans ces deux méthodes, vous pouvez effectuer des changements de couleur de texte et d'arrière-plan. Vous pouvez aussi afficher des images, changer de police de caractères, bref faire tout ce que vous voudrez pour donner à la liste et aux éléments qui la composent l'apparence visuelle que vous voulez à l'affichage ou à la sélection.

Pour utiliser l'interface ListCellRenderer, vous devez créer une classe qui l'hérite et qui

implémente ces deux méthodes. Ensuite, vous allez spécifier au composant List que vous voulez utiliser cette classe comme vue. Cela se fait avec la méthode `setRenderer()` qui prendra en paramètre l'objet de votre classe qui hérite de `ListCellRenderer`. Notez qu'il n'est pas obligatoire d'implémenter les deux méthodes. Tout dépend de vos besoins.

Voici un exemple d'utilisation de `ListCellRenderer`. Nous allons reprendre la liste des pays que nous avons créée à l'[Exemple 2.3](#). Au lieu d'un simple texte comme c'était le cas avec cet exemple, chaque élément de la liste sera maintenant constitué d'un Label qui en plus d'afficher du texte affichera aussi une icône dans son état par défaut. Toujours dans cet état, chaque élément aura une couleur de fond rouge. Quand la sélection passera sur un élément, sa couleur de fond deviendra blanche et quand l'élément sera sélectionné, sa couleur de fond deviendra bleue.

Pour commencer, créons d'abord une classe nommée `Rendu` qui va hériter de la classe `Label` et de l'interface `ListCellRenderer`. Nous allons ensuite ajouter le code des comportements que nous voulons définir dans les deux méthodes héritées de `ListCellRenderer`.

```
public class Rendu extends Label implements ListCellRenderer{  
  
    private Image icone;  
  
    public Rendu()  
    {  
        try{  
            icone=Image.createImage("/arrow.png");  
  
        }catch(IOException ex){ Dialog.show("Erreur", ex.getMessage(), "Ok", null); } }  
    public Component getListCellRendererComponent(List list, Object value, int index,  
        boolean isSelected) { setText(value.toString()); ② setIcon(icone); ③ if(isSelected){  
        getStyle().setBgColor(0x0000ff); ④ } else { getStyle().setBgColor(0xff0000); ⑤ } return this; }  
    public Component getListFocusComponent(List list) {  
        getStyle().setBgColor(0xffffffff); ⑥ return this; } }
```

① Chargement d'une image dans un objet `Image`.

Nous attribuons ici la valeur du paramètre `value` à la méthode `setText()` de `Label` qui **②** permet de définir le contenu d'un texte. Ce paramètre contient le texte d'un élément de la liste.

③ Nous définissons ici l'icône de chaque élément (représenté par le `Label`). Cette icône est l'image que nous avons chargée dans le constructeur de la classe `Rendu`.

④ Ici, nous vérifions la valeur du paramètre `isSelected` et si cette valeur est à `true` alors nous définissons en bleu la couleur de fond du `Label`.

⑤ Si la valeur de ce paramètre `isSelected` est à `false` alors nous définissons la couleur de fond en rouge.

- ⑥ Quand la sélection passera sur un élément de la liste alors définir en blanc sa couleur de fond.

Le code de la création de la liste se modifiera alors en ceci :

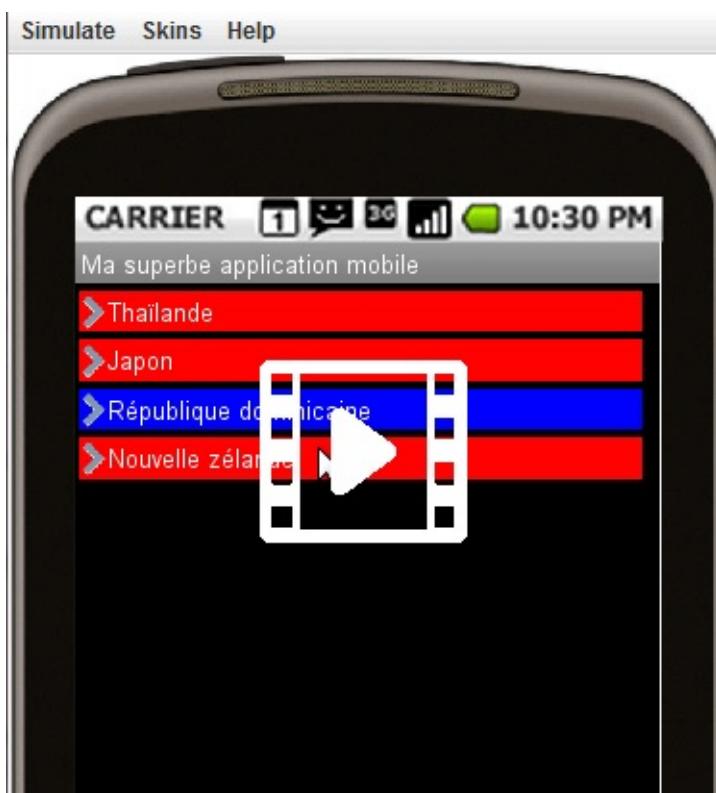
```
//...
DefaultListModel model=new DefaultListModel();
model.addItem("Thaïlande");
model.addItem("Japon");
model.addItem("République dominicaine");
model.addItem("Nouvelle zélande");

List listePays=new List(model);
listePays.setFixedSelection(List.FIXED_NONE_CYCLIC);
listePays.setRenderer(new Rendu());
❶ //...
```

- ❶ Nous faisons appel à la méthode `setRenderer()` pour définir l'objet de la classe qui s'occupera du rendu de la liste.

L'application de notre classe de rendu à la liste donne un résultat qui est celui de la vidéo de la [Figure 2.48](#).

Figure 2.48 : Aperçu d'une liste personnalisée en action (vidéo)



Dans cet exemple, nous avons ajouté des Label à la liste. En fonction de vos besoins, vous pouvez utiliser les composants que vous voulez tout en personnalisant les comportements du mieux que vous pouvez.

Pour plus d'informations sur les styles d'un composant tel qu'utilisé dans cet exemple,

lisez le contenu de la [Section 4.1, Styles d'un composant](#).

Quelques méthodes de List

- `void setSelectedIndex(int index)` – Définit l'élément courant de la liste. En paramètre, l'indice de l'élément à sélectionner.
- `int getSelectedIndex()` – Retourne l'indice de l'élément sélectionné dans la liste. Le premier élément a pour indice 0.
- `Object getSelectedItem()` – Retourne l'élément sélectionné dans la liste.
- `void setOrientation(int orientation)` – Définit l'orientation de la liste. Cette orientation peut être verticale (orientation par défaut) ou horizontale. Les valeurs possibles du paramètre `orientation` sont `List.VERTICAL` ou `List.HORIZONTAL`.
- `ListModel getModel()` – Retourne le modèle utilisé par la liste.
- `int size()` – Retourne le nombre d'éléments dans la liste.

3.21. SwipeableContainer

SwipeableContainer permet de placer un composant sous un autre composant. Ainsi, il suffit de glisser sur la gauche ou sur la droite le composant placé au-dessus pour afficher le contenu en dessous. L'exemple le plus classique et le plus répandu est lorsqu'on fait glisser un bouton sur le côté pour afficher en dessous des options supplémentaires à l'action apportée par le bouton. Dans l'exemple suivant, nous allons placer un CheckBox sous un MultiButton. Ainsi, en glissant le MultiButton vers la droite, nous afficherons le CheckBox.

Exemple 2.4 : Utilisation d'un SwipeableContainer

```
Form f = new Form("Ma superbe application mobile");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
```

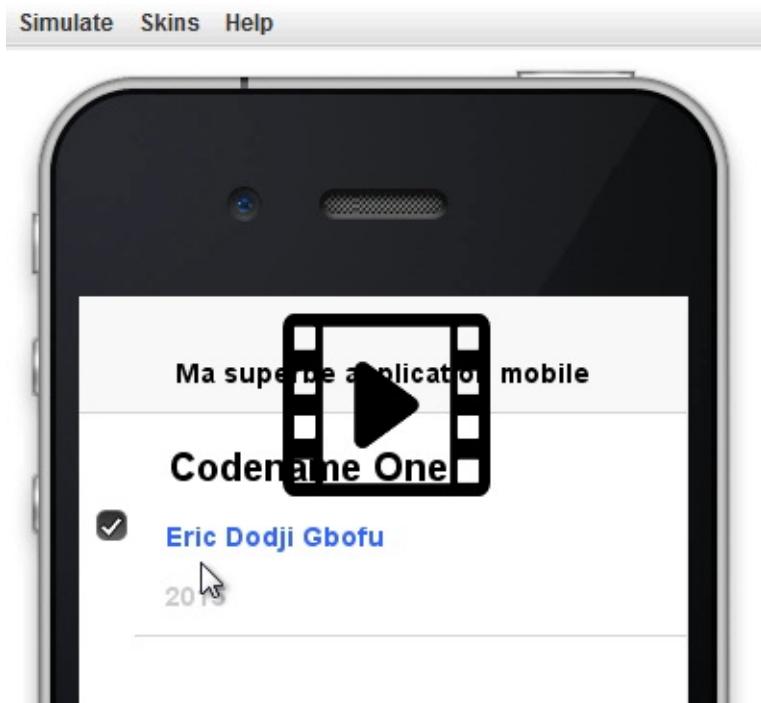
❶ CheckBox choix=new CheckBox(); MultiButton mb=new MultiButton();
mb.setTextLine1("Codename One"); mb.setTextLine2("Eric Dodji Gbofu");
mb.setTextLine3("2015"); SwipeableContainer sc=new SwipeableContainer(choix, mb);
❷ f.addComponent(sc); f.show();

❶ Création d'un CheckBox et d'un MultiButton.

Création de l'objet SwipeableContainer. Le constructeur de cette classe prend en premier paramètre le composant du dessous placé à gauche et en deuxième paramètre ❷ le composant du dessus. Une autre version surchargée du constructeur prend trois

paramètres. Le premier et le troisième sont les mêmes que ceux que nous venons d'utiliser et le deuxième est le composant du dessous placé à droite.

Figure 2.49 : Aperçu d'un SwipeableContainer en action (vidéo)



Quelques méthodes de SwipeableContainer

- `void close()` – Ferme le composant du dessus s'il est ouvert. Cela correspond au fait que ce composant a été glissé sur le côté pour laisser afficher le composant du dessous.
- `boolean isOpen()` – Retourne `true` si le composant du dessus est ouvert. `false` s'il est fermé.
- `void openToLeft()` – Permet de préciser que le composant du dessus peut être ouvert (ou glissé) vers la gauche.
- `void openToRight()` – Permet de préciser que le composant du dessus peut être ouvert (ou glissé) vers la droite.
- `boolean isOpenedToLeft()` – Retourne `true` si le composant du dessus est ouvert sur la gauche.
- `boolean isOpenedToRight()` - Retourne `true` si le composant du dessus est ouvert sur la droite.

3.22. Toolbar

Le Toolbar (traduit en français par *barre d'outils*) est un composant qui permet de personnaliser la barre de titre d'un Form ou d'un Dialog. Ainsi, au lieu d'avoir seulement le titre de la fenêtre sur la barre de titre, vous pouvez avoir le ou les composants que vous voulez à cet endroit. Le Toolbar est étroitement lié à la fonctionnalité du menu hamburger, donc si on lui ajoute des commandes (Command), celles-ci seront automatiquement ajoutées dans un menu hamburger. Voici un exemple complet de la création d'un Toolbar dans lequel nous allons placer une zone de texte et un bouton.

Exemple 2.5 : Création et utilisation d'un Toolbar

```
Form f = new Form();
f.addComponent(new SpanLabel("Hello! Avec le Toolbar, tu peux tout insérer
dans la barre de titre ;-)));
```

❶ Container toolbarC=new Container(new BorderLayout()); TextField zoneRecherche=new TextField(); zoneRecherche.setHint("Rechercher"); Button boutonRecherche=new Button("Ok");
toolbarC.addComponent(BorderLayout.CENTER,zoneRecherche);
toolbarC.addComponent(BorderLayout.EAST,boutonRecherche); Toolbar toolbar=new Toolbar(); ❷ f.setToolBar(toolbar); ❸ toolbar.setTitleComponent(toolbarC); ❹ f.show();

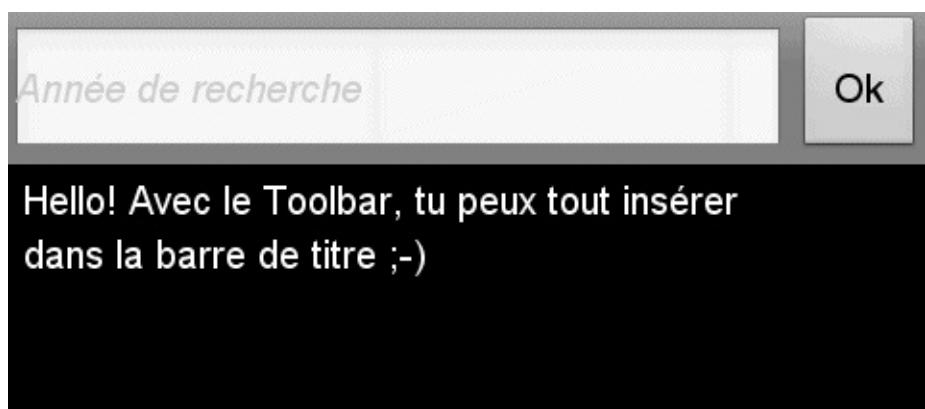
❶ Création d'un conteneur à ajouter au Toolbar. Nous créons aussi une zone de texte et un bouton que nous ajoutons dans le conteneur.

❷ Création de l'objet Toolbar.

❸ Ajout du Toolbar à la fenêtre avec la méthode `setToolBar()` de la classe Form. Cette méthode prend en paramètre l'objet Toolbar créé.

❹ Ajout du conteneur (créé au point ❶) au Toolbar.

Figure 2.50 : Un toolbar contenant une zone de texte et un bouton



Pour ajouter une commande au Toolbar, utilisez la méthode `addCommandToSideMenu()`. Voici un exemple :

```
//...
Command quitter=new Command( "Quitter" );
① Toolbar toolbar=new Toolbar(); f.setToolBar(toolbar);
toolbar.setTitleComponent(toolbarC); toolbar.addCommandToSideMenu(quitter); ②

① Création de la commande Quitter.

② Ajout de cette commande au Toolbar.
```

La [Figure 2.51](#) montre le rendu de ce code. Touchez les trois traits horizontaux pour ouvrir le menu.

Figure 2.51 : Un toolbar associé à un menu hamburger



Quelques méthodes de Toolbar

- void `hideToolbar()` – Masque le Toolbar s'il était affiché.
- void `showToolbar()` – Affiche le Toolbar s'il était masqué.
- void `addComponentToSideMenu(Component cmp)` – Ajoute un composant graphique quelconque au [menu hamburger](#) attaché au Toolbar.

4. Styles et transitions

4.1. Styles d'un composant

L'une des grandes forces de Codename One réside dans la possibilité de personnaliser en profondeur les composants graphiques. Vous pouvez contrôler minutieusement leur apparence en faisant appel à une classe nommée Style. À chaque fois qu'un composant est créé, un objet Style est aussi automatiquement créé par celui-ci. La classe Style vous permet de gérer le visuel des composants à travers les couleurs, les polices de caractères, les marges, la transparence, les images, la disposition à l'écran, les bordures.

Il y a deux manières d'appliquer un style à un composant. L'une est de récupérer l'objet Style qui lui est associé et de faire appel aux méthodes de cette classe pour effectuer ce que vous voulez. Cela se fait avec la méthode `getStyle()` qui se trouve à l'intérieur de chaque composant. L'autre consiste à instancier la classe Style, ajuster les attributs que vous voulez et attribuer l'objet Style instancié au composant concerné avec la méthode `setStyle()`.

Avant de passer à un exemple concret de personnalisation d'un composant, voici quelques méthodes de la classe Style que vous pouvez utiliser en fonction du type d'attribut à modifier. Il en existe beaucoup, donc je ne vous présenterai ici que les principales qui vous aideront à comprendre le fonctionnement de cette classe.

Couleurs et images

- `void setBgColor(int color)` – Permet d'attribuer ou de modifier la couleur de fond d'un composant.
- `void setFgColor(int color)` – Permet d'attribuer ou de modifier la couleur de premier plan. En général, il s'agit de la couleur du texte affiché par le composant.
- `void setBgImage(Image img)` – Permet d'attribuer ou de modifier l'image de fond d'un composant.
- `void setBgTransparency(int transparency)` – Permet de définir le niveau de transparence de l'image de fond d'un composant. La valeur du paramètre doit être comprise entre 0 et 255. 0 étant le niveau de transparence totale, 255 le niveau d'opacité totale.
- `void setBackgroundGradientStartColor(int color)` – Un dégradé étant constitué de deux couleurs, cette méthode permet de définir la première couleur du dégradé.
- `void setBackgroundGradientEndColor(int color)` – Un dégradé étant constitué de deux couleurs, cette méthode permet de définir la deuxième couleur du dégradé.
- `void setBackgroundType(byte backgroundType)` – Définit le type de fond à

appliquer à un composant. Les valeurs possibles du paramètre `backgroundType` sont répertoriées dans le [Tableau 2.2](#). Ces valeurs sont des constantes de la classe `Style`.

Tableau 2.2 : Types d'arrière-plan d'un composant

BACKGROUND_IMAGE_SCALED	Fond avec une image qui sera adaptée à la taille du composant.
BACKGROUND_IMAGE_TILE_BOTH	Fond avec une image qui sera répétée (sous forme de mosaïque) pour remplir le fond du composant.
BACKGROUND_IMAGE_TILE_VERTICAL	Fond avec une image qui sera répétée (sous forme de mosaïque et verticalement) pour remplir le fond d'un composant.
BACKGROUND_IMAGE_TILE_HORIZONTAL	Fond avec une image qui sera répétée (sous forme de mosaïque et horizontalement) pour remplir le fond d'un composant.
BACKGROUND_GRADIENT_LINEAR_HORIZONTAL	Fond dégradé horizontal.
BACKGROUND_GRADIENT_LINEAR_VERTICAL	Fond dégradé vertical.
BACKGROUND_GRADIENT_RADIAL	Fond dégradé en forme de cercle.

Caractères et polices de caractères

- `void setFont(Font f)` – Permet de définir la police de caractères à appliquer au texte d'un composant. En paramètre, un objet de type `Font`.
- `void setTextDecoration(int decoration)` – Permet d'appliquer une décoration (souligné, effet 3D ou autres) au texte d'un composant. Le paramètre `decoration` peut prendre l'une des valeurs du [Tableau 2.3](#). Ces valeurs sont des constantes de la classe `Style`.

Tableau 2.3 : Attributs de textes

TEXT_DECORATION_UNDERLINE	Soulignement en dessous du texte.
TEXT_DECORATION_OVERLINE	Surlignement au dessus du texte.
TEXT_DECORATION_NONE	Pas de décoration.
TEXT_DECORATION_STRIKETHRU	Texte barré.
TEXT_DECORATION_3D	Effet visuel 3D avec ombre sur le texte.

TEXT_DECORATION_3D_SHADOW_NORTH	Effet visuel 3D avec ombre sur le texte. Ici, l'ombre sera placée au dessus du texte.
TEXT_DECORATION_3D_LOWERED	Effet visuel 3D avec un ombrage léger sur le texte.

Alignement, marges, bordures et autres

- `void setAlignment(int align)` – Définit l’alignement d’un composant à l’intérieur d’un Container. Les valeurs possibles du paramètre align sont : Component.CENTER, Component.LEFT, Component.RIGHT.
- `void setMargin(int top, int bottom, int left, int right)` – Définit les marges du style en passant en paramètres les valeurs des marges du dessus, du dessous, de la gauche et de la droite.
- `void setMargin(int orientation, int gap)` – Définit les marges du style. Le premier paramètre orientation définit la zone pour laquelle la marge sera appliquée. Les valeurs possibles de ce paramètre sont : Component.TOP (Haut), Component.BOTTOM (Bas), Component.LEFT (Gauche), Component.RIGHT (Droit).
- `void setPadding(int top, int bottom, int left, int right)` – Permet de définir les marges à l’intérieur du composant contrairement à `setMargin()` qui s’occupe des marges extérieures. En paramètres, les valeurs des marges du dessus, du dessous, de la gauche et de la droite.
- `void setPadding (int orientation, int gap)` – Définit les marges intérieures. Le premier paramètre orientation définit la zone pour laquelle la marge sera appliquée. Les valeurs possibles de ce paramètre sont : Component.TOP (Haut), Component.BOTTOM (Bas), Component.LEFT (Gauche), Component.RIGHT (Droit).
- `void setBorder(Border border)` – Définit les bordures.
- `void setOpacity(int opacity)` – Définit le niveau d’opacité. Le paramètre peut prendre une valeur allant de 0 à 255.

La classe Style offre beaucoup de possibilités à explorer. Un petit tour dans la [Javadoc de cette classe](#) vous aidera à voir tout ce que vous pouvez faire en termes de personnalisation du visuel des composants.

4.2. Transitions

Nous allons maintenant parler d’effets visuels. Les transitions permettent de remplacer un composant par un autre avec un effet d’animation. La plupart du temps, elles sont

appliquées aux composants Form et Dialog. Ainsi, il est possible de définir une animation de transition quand on quitte un Form vers un autre Form (avant le départ du premier et/ou avant l'affichage du second). Même chose avant l'affichage ou la fermeture d'une boîte de dialogue. Codename One propose quelques effets de transition que voici :

- [Slide](#)
- [Fade](#)
- [Cover et Uncover](#)
- [Flip](#)

Ces effets sont disponibles dans la classe CommonTransitions sous forme de méthodes statiques. Cette classe hérite de la classe Transition qu'il est possible d'étendre soi-même dans le but de créer un effet de transition personnalisé, mais nous n'aborderons pas cet aspect. CommonTransitions fonctionne en association avec la classe Motion qui fournit divers modèles de mouvements physiques. Tout comme pour la classe Transition, il est aussi possible d'étendre la classe Motion dans le but de créer d'autres types de mouvements. L'utilisation des transitions par les composants Form et Dialog se fait à l'aide des deux méthodes suivantes :

- `void setTransitionInAnimator(Transition transition)` – Cette méthode exécutera la transition pendant l'affichage du Form ou du Dialog. Elle prend en paramètre la transition à exécuter.
- `void setTransitionOutAnimator(Transition transition)` – Cette méthode exécutera la transition pendant la disparition ou le départ d'un Form ou d'un Dialog pour laisser place à un autre. Elle prend aussi en paramètre la transition à exécuter.

Slide transition

Cette transition permet de créer un effet de défilement de manière horizontale ou verticale.

- `CommonTransitions.createSlide(int type, boolean forward, int speed)`
- `CommonTransitions.createFastSlide(int type, boolean forward, int speed)`

Ces deux méthodes statiques créent un effet de transition de slide. La différence entre les deux est que la seconde méthode est plus axée sur la vitesse d'exécution au détriment de l'utilisation de la mémoire RAM. Ce sont des méthodes surchargées, elles ont donc d'autres équivalents prenant un quatrième paramètre. Le premier paramètre `type` désigne la direction du mouvement de transition qui peut être horizontale ou verticale. Les valeurs possibles de ce paramètre sont `CommonTransitions.SLIDE_HORIZONTAL` et `CommonTransitions.SLIDE_VERTICAL`.

Le second paramètre `forward` définit la direction de départ et de fin du défilement. Si le paramètre `type` est horizontal et que `forward` est à `true`, alors le mouvement de slide se

fera vers la droite. Le mouvement se fera vers la gauche si `forward` est à `false`. Maintenant si `type` est vertical et que `forward` est à `true`, alors le mouvement de slide se fera du haut vers le bas. Et sinon, il se fera vers le haut.

Le troisième paramètre `speed` permet de définir la vitesse d'exécution de la transition en millisecondes.

Fade transition

Cette transition permet de créer un effet de fondu enchaîné. Elle se crée avec la méthode suivante :

- `CommonTransitions.createFade(int duration)`

Le paramètre `duration` permet de définir en millisecondes la durée de la transition.

Cover et Uncover transition

Ces deux transitions sont identiques à cela près que l'une représente l'opposé de l'autre. Elles créent un effet de recouvrement. Ainsi, dans le cas du passage d'un Form à un autre par exemple, le second Form viendra couvrir le premier si la transition utilisée est Cover. En revanche, si c'est Uncover, alors le premier Form s'effacera de manière à dévoiler le second en dessous du premier. Voici les méthodes à utiliser pour créer ces transitions.

- `CommonTransitions.createCover(int type, boolean forward, int speed)`
- `CommonTransitions.createUncover(int type, boolean forward, int speed)`

Leurs paramètres sont les mêmes que ceux des méthodes de création des effets de slide.

Flip transition

Contrairement aux transitions précédentes, celle-ci ne se trouve pas dans la classe `CommonTransitions`. Elle est disponible sous forme de classe indépendante et son utilisation est simple. Pour l'utiliser, il faut passer une instance de la classe `FlipTransition` en paramètre à l'une ou aux méthodes `setTransitionInAnimator()` et `setTransitionOutAnimator()` en fonction de votre besoin.

```
form.setTransitionOutAnimator(new FlipTransition());
```

Exemple : Effets de transition entre quatre pages

```

❶ final Form f = new Form("Page 1"); f.setLayout(new
BoxLayout(BoxLayout.Y_AXIS)); Button b=new Button("Aller à la seconde page");
f.addComponent(b); f.setTransitionOutAnimator(CommonTransitions.createSlide(
CommonTransitions.SLIDE_VERTICAL, true, 1000)); ❷ ❸ final Form f2 = new
Form("Page 2"); f2.setLayout(new BoxLayout(BoxLayout.Y_AXIS)); Button b2=new
Button("Aller à la troisième page"); f2.addComponent(b2);
f2.setTransitionOutAnimator(CommonTransitions.createFade(1000)); ❹ ❺ final Form f3 =
new Form("Page 3"); f3.setLayout(new BoxLayout(BoxLayout.Y_AXIS)); Button
b3=new Button("Aller à la quatrième page"); f3.addComponent(b3);
f3.setTransitionOutAnimator(CommonTransitions.createCover(
CommonTransitions.SLIDE_HORIZONTAL, false, 1000)); ❻ ❻ final Form f4 = new
Form("Page 4"); f4.setLayout(new BoxLayout(BoxLayout.Y_AXIS)); Button b4=new
Button("Retourner à la première page"); f4.addComponent(b4);
f4.setTransitionOutAnimator(new FlipTransition()); ❽ ❾ b.addActionListener(new
ActionListener() { public void actionPerformed(ActionEvent evt) { f2.show(); } });
❿ b2.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent evt) { f3.show(); } });
❿ b3.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent evt) { f4.show(); } });
❿ b4.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent evt) { f.show(); } });
f.show();

```

❶ Création de la première page (avec Form) et d'un bouton pour passer à la page suivante (page 2).

❷ Ajout de l'effet de transition de slide vertical à la première page.

❸ Création de la deuxième page et d'un bouton pour passer à la page suivante (page 3).

❹ Ajout de l'effet de transition de fondu enchaîné à la deuxième page.

❺ Création de la troisième page et d'un bouton pour passer à la page suivante (page 4).

❻ Ajout de l'effet de transition de couverture à la troisième page.

❼ Création de la quatrième page et d'un bouton pour retourner à la première page.

❽ Ajout de l'effet de transition flip à la quatrième page.

❾ Ajout de l'événement de clic au bouton de la page 1. Un clic sur ce bouton affichera la page 2.

❿ Ajout de l'événement de clic au bouton de la page 2. Un clic sur ce bouton affichera la page 3.

⑪ Ajout de l'événement de clic au bouton de la page 3. Un clic sur ce bouton affichera la page 4.

⑫ Ajout de l'événement de clic au bouton de la page 4. Un clic sur ce bouton affichera la page 1.

Figure 2.52 : Rendu des transitions (vidéo)



Persistante des données

Dans ce chapitre, nous parlerons de la persistance de données, donc du stockage. Codename One propose quatre moyens pour stocker les données d'une application : le stockage **brut**, le stockage **dans des fichiers**, le stockage dans **une base de données SQLite** et le stockage **dans le cloud**. Chacun de ces moyens a ses avantages et ses limites et certains sont plus adaptés à certaines situations que d'autres.

1. Avec Storage

La classe Storage fournit un système de stockage simple, portable et fonctionnel sur toutes les plateformes supportées par Codename One. C'est l'équivalent du stockage nommé RMS en programmation J2ME pour ceux qui s'y connaissent. Un stockage fait avec Storage est placé par défaut dans un endroit sécurisé du téléphone. Il est représenté par un fichier dans lequel se trouvent des données brutes de types primitifs et/ou de types objets. Un Storage est propre à chaque application et toutes les informations stockées avec ce moyen seront automatiquement supprimées quand l'application qui les a stockées est désinstallée du téléphone. Storage est une classe singleton donc il n'est pas possible d'avoir plus d'une instance à la fois. Quand on crée un Storage, il faut lui donner un nom avant d'y insérer des données. Ce nom servira à l'ouvrir et à lire les données qui y sont enregistrées.

1.1. Exemples de stockage de données

Exemple 3.1 : Stockage d'une simple donnée

```
String nom="Eric Dodji";  
Storage store=Storage.getInstance();  
① boolean verifEnreg=store.writeObject("infoperso", nom); ② ③ if(verifEnreg==true){  
    Dialog.show("Validation", "Enregistrement effectué", "OK",null); } else {  
    Dialog.show("Validation", "Echec d'enregistrement", "OK",null); }
```

Dans cet exemple, nous récupérons d'abord une instance de Storage **①**. Nous y enregistrons ensuite le contenu de la variable nom. Pour cela, nous utilisons la méthode writeObject(), qui prend en paramètres le nom du Storage et la variable à stocker **②**. writeObject() retourne un booléen pour savoir si l'enregistrement s'est bien effectué ou pas. En fonction de l'état de l'enregistrement, nous affichons une boîte de dialogue avec un message **③**.

Vous pouvez aussi stocker un objet dans un Storage. En voici un exemple.

Exemple 3.2 : Stockage d'un objet

```
Vector individus=new Vector();
```

```
① Hashtable individu=new Hashtable(); individu.put("Nom", "Eric");  
individu.put("Profession", "Programmeur"); individu.put("Age", 28); ②  
individus.addElement(individu); Storage store=Storage.getInstance(); boolean  
verifEnreg=store.writeObject("infoperso", individus); if(verifEnreg==true){  
    Dialog.show("Validation", "Enregistrement effectué", "OK",null); } else {
```

```
Dialog.show("Validation", "Echec d'enregistrement", "OK",null); }
```

Cet exemple stocke dans le Storage nommé *infoperso* un objet de type Vector qui contient un Hashtable ②. Ce Hashtable contient des informations (nom, profession, âge) sur un individu ①.

Voyons maintenant un cas plus complexe de stockage d'objet dans un Storage. Supposons que vous voulez enregistrer des informations de plusieurs individus à la fois dans le Storage en utilisant une classe dédiée pour chacun d'eux. Dans ce cas, nous allons créer notre propre classe, que nous nommerons par exemple Individu, nous l'instancierons, intégrerons les données à l'intérieur puis enregistrerons le tout dans le Storage.

Codename One fournit une interface nommée Externalizable, qui est similaire à l'interface Externalizable de Java SE. Même si elles sont similaires sur certains points, elles sont différentes, donc n'utilisez pas celle de Java SE parce qu'elle n'est pas supportée par l'API de Codename One. Cette interface permet à un objet de se déclarer externalisable. Cela permettra de le sérialiser. Cette sérialisation implique qu'il peut alors être stocké dans un fichier, dans un Storage ou être envoyé à travers un réseau. Avant sa sérialisation, vous devez d'abord l'enregistrer avec une méthode (de la classe Util) que nous verrons plus loin.

La majorité des types et objets manipulés par Codename One sont externalisables. Les types concernés sont les suivants : int[], float[], long[], double[], byte[], Object[], String, Integer, Float, Double, Short, Long, Byte, Character, Boolean, Vector, Hashtable.

Pour commencer, créons une classe Individu, qui va implémenter l'interface Externalizable. Notre classe contiendra trois variables privées (nom, profession, age), les getters et setters de ces variables et trois méthodes de l'interface Externalizable.

Exemple 3.3 : Création d'une classe Individu implémentant la classe Externalizable

```
public class Individu implements Externalizable
{
```

```
① private String nom; private String profession; int age; ② public String getNom() {
return nom; } public void setNom(String nom) { this.nom = nom; } public String
getProfession() { return profession; } public void setProfession(String profession) {
this.profession = profession; } public int getAge() { return age; } public void setAge(int
age) { this.age = age; } ③ public int getVersion() { return 1; } ④ public void
externalize(DataOutputStream out) throws IOException { Util.writeUTF(nom, out);
Util.writeUTF(profession, out); out.writeInt(age); } ⑤ public void internalize(int version,
DataInputStream in) throws IOException { nom=Util.readUTF(in);
profession=Util.readUTF(in); age=in.readInt(); } ⑥ public String getObjectID() { return
"Individu"; } }
```

Déclaration des variables à utiliser pour la récupération des données à enregistrer et à

❶ lire.

❷ Création des getters et setters des variables.

La méthode `getVersion()` retourne la version courante de l'objet. Vous pouvez mettre
❸ la valeur que vous voulez mais sachez que la méthode `internalize()` a besoin de cette valeur.

`externalize()` est la méthode dans laquelle vous devez effectuer les opérations
❹ d'écriture de vos données en vous servant de l'objet `DataOutputStream` présent en paramètre.

`internalize()` est la méthode dans laquelle vous devez effectuer les opérations de
❺ lecture de vos données en vous servant de l'objet `DataInputStream` présent en paramètre.

La méthode `getObjectId()` retourne un identifiant unique représentant notre objet. Cet
❻ identifiant est une chaîne de caractères. Généralement, il est mieux d'utiliser le nom de la classe elle-même.

Note > Les méthodes `writeUTF()` et `readUTF()` font respectivement un travail d'écriture et de lecture des données. Elles s'occupent aussi de l'encodage des données qu'on leur a passées en paramètres.

Nous allons maintenant enregistrer notre classe avec la méthode statique `register()` de la classe Util avant de passer à la sérialisation. Cela se fait comme ceci :

```
Util.register("Individu", Individu.class);
```

Nous pouvons mettre ce code dans la méthode `init()` (de préférence) ou dans la méthode `start()` de la classe contenant la structure de l'application. Nous allons maintenant instancier notre classe Individu puis enregistrer et lire un objet de ce type dans le Storage.

```
Individu ind=new Individu();
ind.setNom("Eric Dodji");
ind.setPrenom("Programmeur");
ind.setAge(28);
Storage.getInstance().writeObject("infosperso", ind);
```

Si vous avez plusieurs objets de type Individu à enregistrer, alors vous pouvez insérer tous ces objets dans un Vector par exemple avant de stocker ce Vector avec `writeObject()`.

1.2. Exemples de lecture de données

```
String nom=(String)Storage.getInstance().readObject("infosperso");
❶ if(nom!=null){ Dialog.show("Salutation", "Hello "+nom, "OK", null); }
```

La méthode `readObject()` permet de lire des données depuis un Storage ①. Elle prend une seule variable en paramètre et retourne un Object. Ce paramètre est le nom du Storage. Étant donné que `readObject()` retourne un Object, il va falloir le convertir à chaque fois vers le type approprié avant de l'utiliser. Dans cet exemple, nous le convertissons vers le type String.

Pour lire un objet de type Individu, tel que nous l'avons créé et stocké précédemment, nous procéderons de la sorte :

```
Individu ind=(Individu)Storage.getInstance().readObject("infosperso");
if(ind!=null){
    String nom=ind.getNom();
    String profession=ind.getProfession();
    int age=ind.getAge();

    Dialog.show("Salutation", "Nom: "+nom+" Profession: "+profession+
Age:"+age, "OK", null);
}
```

Note > Pour rappel, le Storage est la manière la plus portable pour enregistrer des données avec Codename One.

1.3. Quelques méthodes de Storage

- `boolean exists(String name)` – Vérifie si le Storage dont le nom est passé en paramètre existe ou non. Retourne true s'il existe.
- `String[] listEntries()` – Retourne dans un tableau de chaîne de caractères la liste de tous les fichiers de stockage créés par Storage et présents sur le téléphone.
- `void deleteStorageFile(String name)` – Supprime le fichier de Storage dont le nom est passé en paramètre.
- `void clearStorage()` – Supprime de l'appareil tous les fichiers créés par Storage.
- `void clearCache()` – Supprime le contenu du cache. Par défaut, les données de Storage sont mises en cache pour un accès rapide mais cela peut poser des problèmes d'actualisation des données. Supprimer le cache avec cette méthode permettra de recharger à nouveau les données.
- `void setHardCacheSize(int size)` – Attribue une taille au cache.
- `void setNormalizeNames(boolean b)` – Si son paramètre est passé à true alors cette méthode remplacera tous les caractères non valides et présents dans le nom du Storage par le symbole underscore (trait de soulignement ou tiret du huit).

2. Avec Preferences

Si vous voulez fournir des fonctionnalités de sauvegarde des préférences (ou paramètres) dans vos applications, vous pouvez utiliser Storage mais le plus simple est d'utiliser la classe Preferences qui utilise déjà en interne Storage et qui simplifie le stockage. Cette classe est relativement simple à utiliser et vous n'aurez pas besoin de l'instancier avant de vous en servir parce qu'elle est constituée uniquement de méthodes statiques.

La classe Preferences fournit deux méthodes surchargées nommées `get()` et `set()`. La méthode `set()` pour ajouter des données et la méthode `get()` pour récupérer les données.

Attention > *La classe Preferences de Codename One est différente de celle de Java SE, donc ne confondez pas les deux et n'incluez pas le package de ce dernier.*

2.1. Exemple : Stockage et lecture

```
Preferences.set("nom", "Eric");
```

❶ String nom=Preferences.get("nom", null); ❷

Le premier paramètre des méthodes `set()` et `get()` est le nom de la donnée à stocker ou à lire. En deuxième paramètre de `set()`, la valeur de la donnée à stocker. Ainsi, sur la première ligne de l'exemple, on stocke le nom *Eric* dans la variable nommée *nom* ❶. Le deuxième paramètre de la méthode `get()` est la valeur que cette méthode doit retourner par défaut si elle ne trouve aucune donnée à lire. Dans notre exemple, on demande à `get()` de retourner une valeur `null` ❷. Si elle en trouve, alors elle retournera ce qui a été trouvé.

Le deuxième paramètre des méthodes `set()` et `get()` ne peut accepter que l'un des types suivants : `int`, `float`, `long`, `double`, `boolean`, `String`.

2.2. Quelques méthodes de Preferences

- `void delete(String pref)` – Supprime un élément ajouté au Storage créé par la classe Preferences. En paramètre, le nom de l'élément à supprimer.
- `void clearAll()` – Supprime tous les éléments ajoutés au Storage créé par la classe Preferences.

3. Avec Database (pour les bases de données SQLite)

De nos jours, les smartphones peuvent contenir une base de données, mais pas n'importe laquelle. Il s'agit de SQLite. SQLite est super légère et tient dans un seul fichier. Elle n'est pas supportée par toutes les plateformes de smartphones. Parmi celles supportées par Codename One, elle n'est disponible que sur iOS, Android et BlackBerry OS (version 5 à 7). Si vous visez la portabilité de vos données sans souci quelle que soit la plateforme alors utiliser une base de données n'est pas forcément le meilleur choix. En revanche, je la conseillerais si vous avez une grande quantité de données à gérer ou des données complexes à manipuler.

La classe qui gère les bases de données SQLite en Codename One se nomme Database.

3.1. Requête simple

```
Database db=Database.openOrCreate("clients.db");
```

❶ db.execute("insert into clients values('Eric','Programmeur')"); ❷ db.close(); ❸

Cet exemple ouvre une base de données SQLite nommée clients.db avec openOrCreate() ❶, exécute une requête SQL d'insertion avec execute() ❷ puis ferme la base de données avec close() ❸.

La méthode openOrCreate() permet d'ouvrir le fichier de la base de données s'il existe. S'il n'est pas trouvé alors il sera créé.

La méthode execute() permet d'exécuter une requête SQL d'insertion ou de mise à jour. Pour une requête SQL d'extraction de données comme c'est le cas avec la commande SQL SELECT, il faut plutôt utiliser la méthode executeQuery().

3.2. Exemple d'utilisation

Voici un exemple basique mais complet, avec une requête d'insertion et d'extraction de données. En résumé, il faut saisir dans deux zones de texte un nom et une profession puis appuyer sur l'un des deux boutons disponibles (voir Figure 3.1). L'un permet d'enregistrer les données entrées dans une base de données SQLite et l'autre d'afficher dans la console de votre éditeur de code la liste de tous les enregistrements de la base de données.

```
public class TestBaseSQLite {  
  
    private Database db;  
  
    ❶ public void start() { chargerCreerBase(); ❷ Form f = new Form("Ma superbe  
application mobile"); f.setLayout(new BoxLayout(BoxLayout.Y_AXIS)); ❸ final
```

```

TextField nom=new TextField(); nom.setHint("Entrez votre nom"); final TextField
profession=new TextField(); profession.setHint("Entrez votre profession"); Button
enregistrer=new Button("Enregistrer"); ❸ enregistrer.addActionListener(new
ActionListener() { public void actionPerformed(ActionEvent evt) { try { ❹
db.execute("insert into employees
values'" +nom.getText()+"','"+profession.getText()+"')"); nom.clear();
profession.clear(); } catch (IOException ex) { Dialog.show("Erreur de requête",
ex.getMessage(), "OK", null); } } });
Button afficher=new Button("Afficher"); ❺ afficher.addActionListener(new ActionListener() { public void
actionPerformed(ActionEvent evt) { try { ❻ Cursor c=db.executeQuery("select * from
employees"); while(c.next()){ Row ligne=c.getRow(); System.out.println("Nom:
"+ligne.getString(0)+", Profession: "+ligne.getString(1)); } } catch (IOException ex) {
Dialog.show("Erreur de requête", ex.getMessage(), "OK", null); } } });
Container cBut=new Container(new FlowLayout(Component.CENTER));
cBut.addComponent(enregistrer); cBut.addComponent(afficher); f.addComponent(nom);
f.addComponent(profession); f.addComponent(cBut); f.show(); } private void
chargerCreerBase() { try { db=Database.openOrCreate("clients.db"); ❻ if(db==null){
Dialog.show("Incompatible", "SQLite n'est pas supportée par votre
plateforme.", "OK",null); return; } ❻ boolean exists=Database.exists("clients.db");
if(exists==true) { db.execute("CREATE TABLE IF NOT EXISTS [employees] (\n" +
"[nom] VARCHAR(40) NOT NULL, \n" + "[profession] VARCHAR(255) NOT
NULL);"); } } catch (IOException ex) { Dialog.show("Erreur de chargement",
ex.getMessage(), "OK", null); } } }

```

❶ Déclaration de l'objet Database.

❷ Appel de la fonction chargerCreerBase() qui charge la base ou qui la crée si elle n'existe pas.

❸ Création de deux zones de texte pour recueillir le nom et la profession. Création d'un bouton pour enregistrer les informations qui seront entrées par l'utilisateur.

❹ Gestion de l'événement de clic sur le bouton enregistrer.

Exécution de la requête d'insertion des données saisies par l'utilisateur. L'insertion est suivie de la suppression des données entrées dans les champs nom et profession. Pour exécuter cette requête, nous utilisons la méthode execute() qui prend en paramètre la requête à exécuter.

❺ **Astuce** > Pour ne pas avoir à concaténer dans la requête les valeurs à entrer dans la base de données comme c'est fait sur cette ligne, il est possible d'utiliser une autre version surchargée de execute() comme suit : `db.execute("insert into employees values(?,?)", new String[]{ligne.getString(0),ligne.getString(1)});`

❻ Gestion de l'événement de clic sur le bouton afficher.

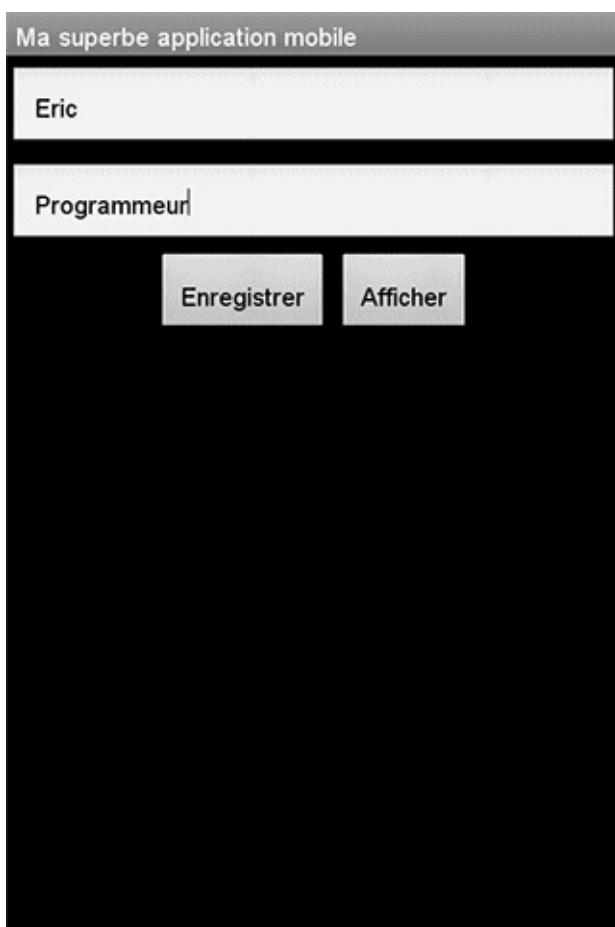
Exécution de la requête d'extraction de données suivie de l'affichage de ces données dans la console. Ici, nous utilisons la méthode `executeQuery()` (à utiliser seulement pour les requêtes `SELECT`). Cette méthode retourne un objet de type `Cursor`. `Cursor` nous permettra de parcourir les résultats de la requête. Pour cela, nous créons une

⑦ boucle `while` puis nous faisons appel à la méthode `next()` pour déplacer le curseur sur la prochaine ligne de résultats. Dans cette boucle, nous appelons la méthode `getRow()` de `Cursor` pour retourner un objet `Row`. `Row` représente une ligne de la base de données. Ainsi avec `Row`, nous pouvons avoir accès aux données de la première et deuxième cellule de la ligne retournée avec la méthode `getString()`. Nous utilisons `getString()` parce que `String` est le type des données que nous voulons récupérer.

Nous voici dans la méthode `chargerCreerBase()` appelée à la ligne ②. Ici nous créons ⑧ l'objet `Database` puis nous ouvrons la base de données `clients.db` si elle existe. Si elle n'existe pas alors elle sera créée. Les deux actions sont effectuées par `openOrCreate()`.

Vérification de l'existence de la base de données `clients.db` avec la méthode ⑨ `exists()`. Si elle existe alors la requête SQL de création d'une table (nommée `employees`) sera exécutée.

Figure 3.1 : Formulaire de saisie



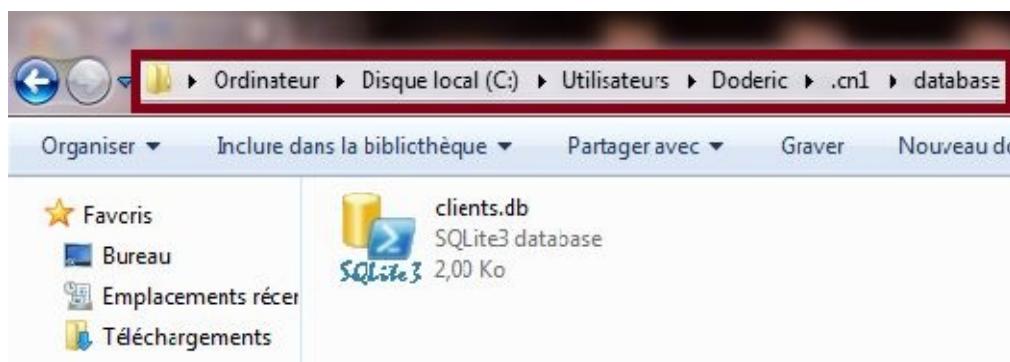
La sortie console des données de la base de données ressemblera à ceci :

Nom: Eric , Profession: Programmeur

3.3. Quelques méthodes de Database

- `void delete(String dbName)` – Supprime la base de données dont le nom est paramètre.
- `void execute(String sql)` – Exécute une requête de mise à jour (`INSERT`, `UPDATE`, `DELETE`, `CREATE`). Le paramètre `sql` est la requête SQL à exécuter.
- `void execute(String sql, Object[] params)` – Exécute une requête de mise à jour (`INSERT`, `UPDATE`, `DELETE`, `CREATE`). Le premier paramètre est la requête SQL à exécuter. Cette requête peut être construite avec le signe du point d'interrogation `(?)`. Le second paramètre fournira les données qui remplaceront les points d'interrogations `(?)` placés dans la requête.
- `Cursor executeQuery(String sql)` – Exécute une requête SQL d'extraction de données (`SELECT`).
- `void beginTransaction()` – Lance l'exécution d'une transaction.
- `void commitTransaction()` – Valide une transaction.
- `void rollbackTransaction()` – Annule la dernière transaction effectuée.
- `String getDatabasePath(String dbName)` – Retourne le chemin vers le fichier de la base de données dont le nom est passé en paramètre.
- `void close()` – Ferme la base de données.

Astuce > À l'attention des utilisateurs de PC sous Windows : Vous pouvez retrouver l'emplacement de la base de données SQLite (créée par le simulateur) sur votre ordinateur dans le dossier *database* du dossier `.cn1` qui se trouve dans le dossier portant le nom de votre session Windows. Ce dernier se trouve dans le dossier *Utilisateurs* qui se trouve sur le disque contenant votre installation de Windows. Sur mon ordinateur, ça donne ceci (voir le chemin dans la zone encadrée) :



4. Avec FileSystemStorage (pour les fichiers)

FileSystemStorage est la classe qui permet la manipulation des fichiers. Vous pouvez l'utiliser pour écrire dans un fichier ou pour y lire des données. Elle peut être comparée à la classe File de Java SE.

4.1. Exemple

Dans l'exemple suivant, nous allons créer un fichier de format CSV et y enregistrer quelques informations.

```
try {
    FileSystemStorage fs=FileSystemStorage.getInstance();

❶ OutputStream stream=fs.openOutputStream("clients.csv");
❷ ❸ String entete="Nom;Profession;\n";
    String infos="Eric;Programmeur;\n";
❹ stream.write(entete.getBytes());
    stream.write(infos.getBytes());
❺ stream.close();
}

catch(IOException ex){
    Dialog.show("Erreur", ex.getMessage(), "Ok", null);
}
```

❶ Création de l'objet FileSystemStorage.

❷ Création d'un flux de sortie (OutputStream) pour la création du fichier clients.csv avec la méthode openOutputStream(). Pour ouvrir le fichier en lecture, utiliser la méthode openInputStream() qui retournera un objet de type InputStream. InputStream fournit les méthodes de lecture.

❸ Création des données à stocker dans le fichier CSV.

❹ Écriture des données dans le fichier avec le flux créé.

❺ Fermeture du flux.

L'exécution de ce code va créer dans le dossier du projet le fichier clients.csv, dont le contenu ressemble à ceci :

Figure 3.2 : Contenu du fichier clients.csv sous Microsoft Excel

	A1	B	C	D
1	Nom	Profession		
2	Eric	Programmeur		
3				

Attention > Cet exemple s'exécutera avec le simulateur sur votre ordinateur, mais pas sur le téléphone. La raison en est qu'aucun chemin n'a été spécifié. En mettant directement le

nom du fichier dans la méthode `openOutputStream()`, vous n'indiquez aucun emplacement sur le téléphone. Pour résoudre ce problème, voici une fonction qui vous fournira la racine de l'endroit où vous pouvez enregistrer un fichier.

```
public String recupererRacine()
{
    String racine=null;

❶ FileSystemStorage fs= FileSystemStorage.getInstance(); String[] racines=fs.getRoots();
❷ int nbreRacine=racines.length; ❸ for(int i=0;i<nbreRacine;i++){ ❹
if(fs.getRootType(racines[i])==FileSystemStorage.ROOT_TYPE_SDCARD) ❺ {
    racine=racines[i]; ❻ if(!racine.endsWith("/")){ racine+="/"; } } break; } } return racine; }
```

- ❶ Déclaration de la variable `racine`. Cette variable contiendra la racine de l'espace de stockage à utiliser.
- ❷ Récupération des espaces de stockage disponibles sur l'appareil dans un tableau de chaînes de caractères.
- ❸ Récupération du nombre total d'espaces de stockage trouvés.

Exécution d'une boucle de recherche dans le but de trouver l'espace de stockage qui correspondra à celui d'une carte mémoire (`ROOT_TYPE_SDCARD`). Pour utiliser l'espace interne de l'appareil, utilisez la constante `ROOT_TYPE_MAINSTORAGE`. Pour le test avec le simulateur sur l'ordinateur, `getRootType()` pourrait retourner la valeur précédente ou la valeur `ROOT_TYPE_UNKNOWN` donc pensez à adapter le code à ce niveau quand vous ❹ êtes sur ordinateur parce que la valeur `ROOT_TYPE_SDCARD` de notre fonction ne retournera aucun emplacement valide sur PC.

Attention > Même s'il est possible d'utiliser l'espace interne de l'appareil avec `ROOT_TYPE_MAINSTORAGE`, je le déconseille parce que beaucoup de constructeurs verrouillent cet espace en interdisant l'écriture de données. Dans ce cas, vous aurez un `NullPointerException` à l'exécution.

Nous essayons de trouver lequel des espaces de stockage retourné par `getRoots()` ❺ correspond à celui d'une carte mémoire. Une fois trouvé, nous attribuons sa valeur à la variable `racine` et nous sortons de la boucle avec `break` pour stopper la recherche.

Sur cette ligne, nous vérifions si le contenu de la variable `racine` se termine par le ❻ caractère slash (“/”). Si ce n'est pas le cas alors nous concaténons ce caractère à la fin de cette variable.

L'introduction de cette fonction nous pousse à modifier une partie de notre code comme suit :

```
String racine=recupererRacine();
```

```
❶ FileSystemStorage fs=FileSystemStorage.getInstance(); OutputStream
```

```
stream=fs.openOutputStream(racine+“clients.csv”); ②
```

① Récupération de la racine de la carte mémoire du téléphone dans la variable racine.

② Concaténation de la variable racine au nom du fichier dans la méthode `openOutputStream()`. Cela nous permet d'avoir le chemin complet du fichier.

Note > Au lieu d'utiliser la fonction `recupererRacine()` pour récupérer un emplacement, vous pouvez aussi utiliser la méthode `getAppHomePath()` qui retourne le chemin vers le dossier de l'application sur l'appareil. Tout comme le chemin renvoyé par `recupererRacine()`, vous devez aussi concaténer le nom du fichier à enregistrer au chemin renvoyé par `getAppHomePath()`.

4.2. Quelques méthodes de FileSystemStorage

- `boolean exists(String file)` – Indique si le fichier dont le nom est passé en paramètre existe. Retourne la valeur `true` si c'est le cas et `false` dans le cas contraire.
- `void delete(String file)` – Supprime le fichier dont le nom est passé en paramètre.
- `long getLength(String file)` – Retourne la taille du fichier dont le nom est passé en paramètre.
- `boolean isDirectory(String file)` – Indique si le fichier dont le nom est passé en paramètre est un dossier.
- `String[] listFiles(String dossier)` – Retourne un tableau de chaînes de caractères contenant la liste de tous les fichiers qui se trouvent dans le dossier dont le nom est passé en paramètre. Les noms des fichiers ne contiendront pas le chemin complet. Il s'agit du chemin relatif et non absolu.
- `void mkdir(String dossier)` – Permet de créer le dossier dont le nom est passé en paramètre.
- `void rename(String fichier, String nouveauNom)` – Permet de renommer un fichier existant. Placez en premier paramètre le nom actuel du fichier et en deuxième paramètre le nouveau nom du fichier.
- `char getFileSystemSeparator()` – Retourne le caractère utilisé pour les chemins de fichiers sur la plateforme sur laquelle s'exécute l'application. Généralement, il s'agit du caractère slash “/”. Utilisez cette méthode pour définir un chemin quand vous n'êtes pas certain du caractère qu'utilisent les plateformes visées par vos applications.
- `String getAppHomePath()` – Retourne le dossier principal de l'application. Elle est pratique pour récupérer un chemin pour le stockage de fichiers.
- `String[] getRoots()` – Retourne dans un tableau de chaînes de caractères tous les

emplacements (ou zones) de stockage disponibles sur la plateforme.

- `int getRootType(String root)` – Retourne le type de la zone de stockage passée en paramètre. Les valeurs possibles de ce type sont : `ROOT_TYPE_MAINSTORAGE` (représente l'espace de stockage intégré au téléphone ; sur ordinateur, c'est le disque dur principal), `ROOT_TYPE_SDCARD` (représente l'espace de stockage d'une carte mémoire), `ROOT_TYPE_UNKNOWN` (représente un espace de stockage inconnu et différent des deux précédents).
- `long getRootAvailableSpace(String root)` – Retourne l'espace disponible sur la zone de stockage passée en paramètre.
- `long getRootSizeBytes(String root)` – Retourne l'espace total de la zone de stockage passée en paramètre.

5. Avec CloudStorage (pour le cloud)

Le serveur cloud de Codename One ne permet pas seulement de compiler les applications, il offre aussi un espace pour stocker des données et des fichiers. Cependant, pour jouir pleinement de ce service sans aucune limitation, il faut être un abonné payant de Codename One. Vous pouvez l'utiliser avec un compte gratuit, mais vous serez limité au niveau de la taille des données à stocker : vous ne pourrez pas stocker plus de dix mille objets ni des fichiers dont la totalité des tailles dépassent 2 Mo. Avant de passer à un exemple, voici quelques explications complémentaires sur le fonctionnement de la classe CloudStorage.

CloudStorage permet de stocker des objets (et des fichiers) dans le cloud de Codename One, mais il ne s'agit pas de n'importe quel type d'objet. Le seul objet qu'il est possible de stocker est de type CloudObject qui est un type propre à Codename One. Ainsi pour stocker des données dans ce cloud, vous devez d'abord créer un objet de ce type où vous mettrez ensuite des informations de n'importe quel autre type. CloudObject stocke les données sous forme de clé-valeur et fournit des getters et setters pour ajouter, modifier et récupérer les données. Enfin, vous devez lui attribuer un niveau de visibilité (voir [plus loin](#) dans l'exemple qui suit).

Attention > *Il est conseillé que la taille d'un objet CloudObject ne dépasse pas 100 Ko pour éviter que le serveur ne retourne une erreur. Ceci est une limitation délibérée de la part des concepteurs de Codename One. Si vous avez besoin de stocker des données volumineuses, alors optez pour l'utilisation d'un fichier.*

Passons maintenant à un exemple.

5.1. Exemple

```
Form f = new Form("Ma superbe application mobile");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

final TextField nom=new TextField();
nom.setHint("Entre ton nom beau gosse");

final TextField profession=new TextField();
profession.setHint("Entre ta profession aussi");

Button enreg=new Button("Enregistrer dans le cloud");
enreg.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        CloudObject cObj=new CloudObject("Identite",
        CloudObject.ACCESS_PUBLIC_READ_ONLY);

❶❷ cObj.setString("nom", nom.getText()); cObj.setString("profession",
profession.getText()); ❸ cObj.setIndexString(1, profession.getText());
CloudStorage.getInstance().save(cObj); ❹ int
```

```

resultat=CloudStorage.getInstance().commit(); ⑤
if(resultat!=CloudStorage.RETURN_CODE_SUCCESS){ ⑥ Dialog.show("Erreur",
"Erreur d'enregistrement "+resultat, "Ok", null); } else { Dialog.show("Effectué",
"Données envoyées dans le cloud", "Ok", null); } });
Button afficher=new
Button("Afficher contenu du cloud"); afficher.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) { try{ CloudObject[]
elements=CloudStorage.getInstance().querySorted("Identites", 1, true, 0, 5,
CloudObject.ACCESS_PUBLIC_READ_ONLY); ⑦ ⑧ for(CloudObject cObj:elements){
Dialog.show("Identite", "Bonjour "+cObj.getString("nom") + " le
"+cObj.getString("profession"), "Ok", null); } } catch (CloudException ex){
Dialog.show("Erreur", "Erreur de lecture "+ex.getMessage(), "Ok", null); } } });
f.addComponent(nom); f.addComponent(profession); f.addComponent(enreg);
f.addComponent(afficher); f.show();

```

Sur cette ligne, nous créons un objet de type CloudObject. CloudObject a trois constructeurs et celui que nous utilisons prend deux paramètres. Le premier paramètre est nommé *type*. Considérez celui-ci comme le nom de l'objet dans lequel vous allez stocker vos données sur le serveur. Le second paramètre définit le niveau de visibilité de vos données. Ce niveau de visibilité peut prendre l'une des cinq valeurs suivantes :

Tableau 3.1 : Niveaux de visibilité d'un CloudObject

ACCESS_PUBLIC	L'objet est publiquement accessible dans le cloud et pourra être modifié par n'importe qui.
ACCESS_PUBLIC_READ_ONLY	L'objet est publiquement accessible dans le cloud mais uniquement en lecture seule. Il ne pourra être modifié que par son créateur.
ACCESS_PRIVATE	L'objet n'est accessible qu'à son créateur et ne pourra être modifié que par lui.
ACCESS_APPLICATION	L'objet est accessible uniquement en lecture par d'autres applications. La modification n'est possible que par son créateur.
ACCESS_APPLICATION_READ_ONLY	L'objet est accessible et modifiable par n'importe quelle application.

Récupération et ajout du nom et de la profession saisis par l'utilisateur dans l'objet CloudObject créé à l'étape précédente. Ici, les ajouts se font avec la méthode *setString()* parce que les données que nous voulons stocker sont des chaînes de caractères. D'autres méthodes de ce genre sont disponibles pour stocker d'autres types ② de données comme *setInteger()* pour une variable de type *int* et *setBoolean()* pour un objet de type *boolean*. La méthode *setString()* a deux paramètres. Le premier est le nom de la clé (un nom pour représenter la donnée à stocker) et le second est la valeur de la clé.

Nous ajoutons ici la profession de l'utilisateur à l'index 1 de notre objet. Cet ajout se fait avec `setIndexString()`. Les équivalents de cette méthode pour d'autres types sont aussi disponibles. La longueur maximale de la chaîne de caractères acceptée est

- ③ 500 octets. Les index permettent d'accélérer les recherches. Dix index sont prévus par objet et vous pouvez y insérer les données à rechercher et à trier. Dans cet exemple, nous ajoutons la profession de l'utilisateur.

Ici, nous récupérons une instance de la classe singleton `CoudStorage` et nous faisons

- ④ appel à la méthode `save()` pour enregistrer notre objet `CloudObject` dans une file d'attente dont le contenu sera envoyé après sur le serveur. Attention, il ne s'agit pas encore de l'envoi proprement dit des données vers le cloud.

Envoi des données sur le serveur cloud de Codename One avec la méthode `commit()`. Cette méthode existe en deux versions. Celle utilisée dans cet exemple n'est pas asynchrone. `commit()` retourne une valeur pour signaler si l'envoi des données a été bien effectué ou pas. La valeur renournée par cette méthode peut prendre l'une des valeurs suivantes :

Tableau 3.2 : Valeurs de retour de la méthode commit()

RETURN_CODE_SUCCESS	L'opération d'envoi s'est bien déroulée.
RETURN_CODE_FAIL_SERVER_ERROR	Une erreur s'est produite au niveau du serveur.
⑤ RETURN_CODE_FAIL_QUOTA_EXCEED	Le quota de stockage fourni a été dépassé. En gros, l'espace fourni est déjà plein.
RETURN_CODE_FAIL_PERMISSION_VIOLATION	Il y a une violation de permission comme essayer de lire par exemple des données privées non accessibles publiquement.
RETURN_CODE_SUCCESS_OBJECT_MODIFIED	L'objet a été modifié au niveau du serveur.
RETURN_CODE_SUCCESS_EMPTY_QUEUE	Valeur indiquant que la file d'attente est vide.

Vérification de la valeur renournée par `commit()`. Que l'envoi soit bien effectué ou pas, une boîte de dialogue avec un message s'affichera (voir [Figure 3.3](#)).

Figure 3.3 : Confirmation d'envoi des données dans le cloud



Nous sommes dans le code de l'action qui sera effectuée quand on appuiera sur le bouton Afficher contenu du cloud. Sur cette ligne, nous faisons appel à la méthode querySorted() pour exécuter une requête d'extraction de données. Une version asynchrone de cette méthode existe.

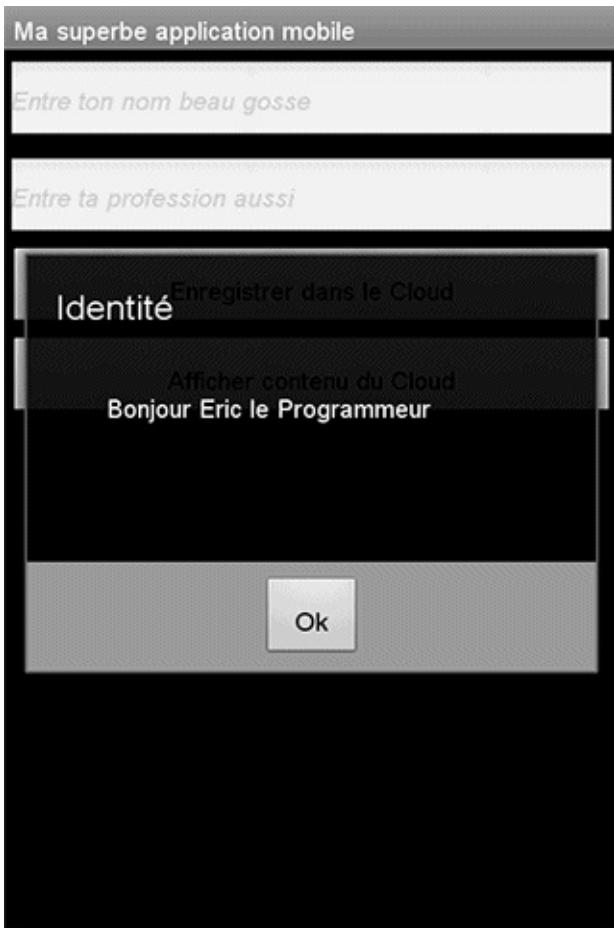
Les différents paramètres de querySorted() sont les suivants. En premier paramètre, on entre le nom de l'objet à interroger dans le cloud. Le second paramètre désigne l'index sur lequel vous voulez effectuer votre recherche. Ici, nous choisissons l'index 1

7 qui est d'ailleurs le seul index que nous utilisons sur les dix disponibles. Le troisième paramètre permet de préciser si on veut recueillir les données dans un ordre croissant ou décroissant. La valeur true placée à cet endroit veut dire qu'on choisit l'ordre croissant. Les paramètres quatre et cinq veulent dire que nous voulons seulement récupérer les données des cinq premiers objets enregistrés (paramètre cinq) en commençant par l'objet de la position zéro (paramètre quatre). Le dernier paramètre est réservé au niveau de visibilité de l'objet. La valeur à placer à cet endroit est semblable à celle passée en paramètre au constructeur de CloudObject au point 1.

Étant donné que querySorted() retourne un tableau de CloudObject, nous parcourons alors ce tableau avec une boucle pour récupérer le nom et la profession de chaque

8 CloudObject récupéré et nous affichons ces informations avec une boîte de dialogue (voir Figure 3.4). À l'inverse de setString() utilisée à la ligne 2, getString() retourne la valeur de la clé passée en paramètre. querySorted() possède aussi une version asynchrone.

Figure 3.4 : Affichage des données lues dans le cloud



Comme mentionné dans le texte d'introduction de cette section, il est aussi possible d'envoyer les fichiers (en plus des données brutes comme nous l'avons vu) dans le cloud de Codename One. Cela se fait avec la méthode `uploadCloudFile()` dont voici le prototype

```
String uploadCloudFile(String typeMime, String file);
```

Cette méthode prend le type MIME du fichier en premier paramètre et l'adresse du fichier à uploader dans le cloud en second paramètre. `uploadCloudFile()` retourne une chaîne de caractères qui représente l'ID unique de ce fichier dans le cloud. Cet ID pourra par exemple servir à supprimer le fichier du cloud.

5.2. Cloud Objects Viewer

Le Cloud Objects Viewer est un petit programme intégré au simulateur de Codename One et qui permet de visualiser les données enregistrées dans le cloud avec CloudStorage. Vous pouvez aussi l'utiliser pour effectuer des requêtes sur ces objets. Pour ouvrir ce programme, allez dans le menu Simulate du simulateur et cliquez sur Cloud Objects Viewer (voir l'encadré de la [Figure 3.5](#)). La [Figure 3.6](#) montre l'interface du programme.

Figure 3.5 : Accès à Cloud Objects Viewer avec le simulateur

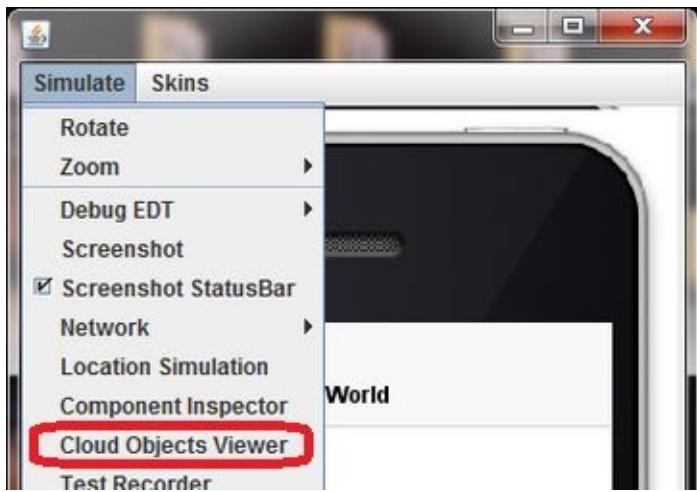
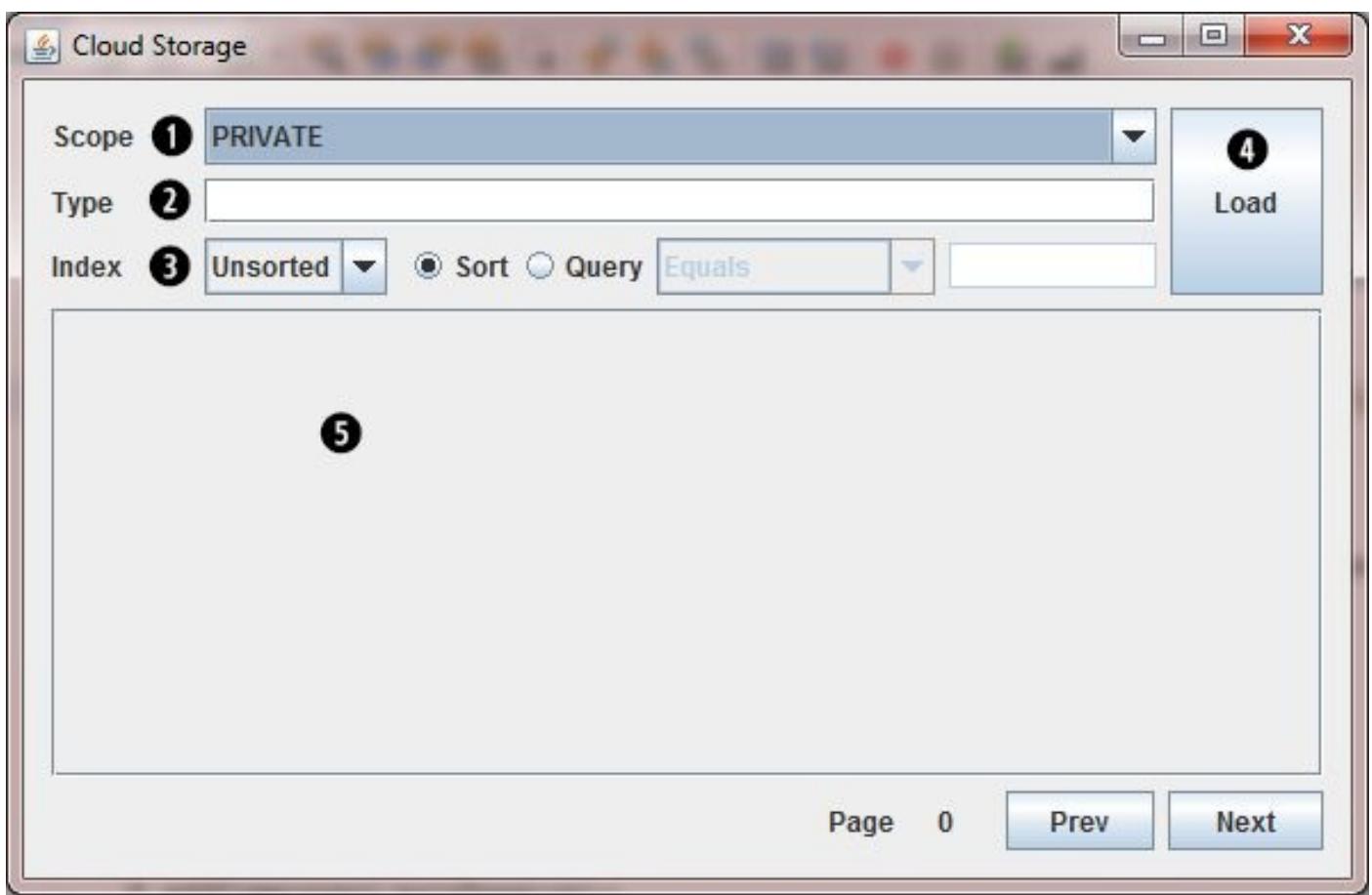


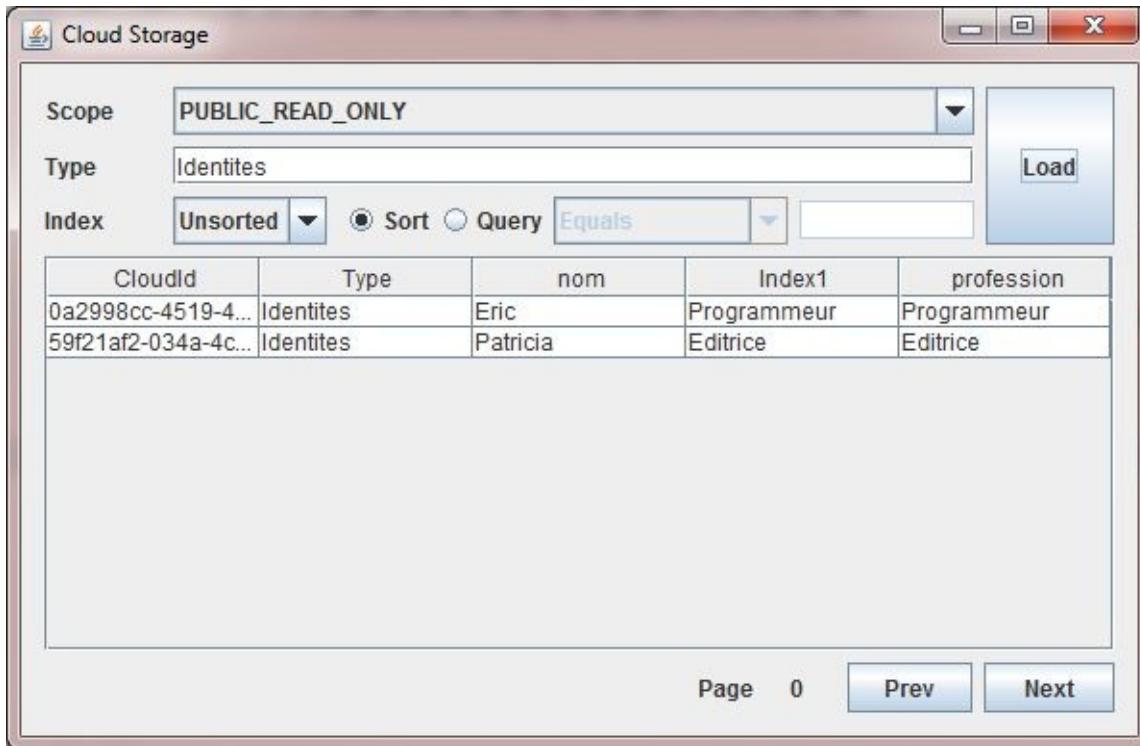
Figure 3.6 : Interface de l'application Cloud Objects Viewer



Le champ Scope ① permet de choisir l'un des différents niveaux de visibilité disponibles. Le champ Type ② recevra le nom de l'objet CloudObject à interroger. Le champ Index ③ propose des options de tri et de requête et le bouton Load ④ permet de lancer le chargement des données qui seront affichées dans la zone ⑤.

Nous allons maintenant tester l'affichage des données que nous avions enregistrées dans le cloud avec le Cloud Objects Viewer. Pour rappel, vous devez être connecté à Internet pour effectuer ce test. Dans la zone Scope de la fenêtre, choisissez PUBLIC_READ_ONLY et dans la zone Type, entrez *Identites*. Pour finir, cliquez sur le bouton Load pour charger et afficher les données (voir la Figure 3.7).

Figure 3.7 : Utilisation de Cloud Objects Viewer pour l'affichage des données du cloud



5.3. Quelques méthodes de CloudStorage

- `void delete(CloudObject obj)` – Supprime du cloud l'objet passé en paramètre.
- `boolean deleteCloudFile(String idFichier)` – Supprime du cloud le fichier dont l'ID est passé en paramètre. Elle retourne true si la suppression s'est bien effectuée.
- `String getUrlForCloudFileId(String idFichier)` – Convertit l'ID du fichier passé en paramètre en une adresse pour permettre le téléchargement du fichier depuis le cloud.
- `void rollback()` – Annule la dernière action effectuée.
- `CloudObject[] fetch(String[] cloudIds)` – Parcourt les objets dont les ID se trouvent dans le tableau d'ID des objets passés en paramètre. Un tableau de CloudObject est retourné.
- `int refresh(CloudObject[] objects)` – Utilise les données disponibles sur le serveur cloud pour mettre à jour les objets passés en paramètre. Cela se fera uniquement si les données en ligne ont été modifiées.
- `CloudObject[] queryEquals(String type, int index, String valeur, int page, int limite, int visibilite)` – Exécute une requête pour trouver et récupérer les objets dont les clés correspondent à la valeur passée en troisième paramètre. Les autres paramètres sont les mêmes que ceux de la méthode `querySorted()` détaillée au point ⑦ de l'exemple vu précédemment.
- `void commit(CloudResponse<Integer> response)` – Cette méthode est une version

asynchrone du `commit()` que nous avons vu dans l'exemple au point ❶. Elle prend en paramètre l'interface `CloudResponse` qui vous fournit deux méthodes à implémenter. La première est `onSuccess()`, qui sera appelée si l'envoi de vos données est bien effectué, et la seconde est `onError()`, qui sera appelée en cas d'échec d'envoi.

Multimédia (photo, audio, vidéo)

Dans ce chapitre, nous allons aborder les notions de médias à savoir apprendre à utiliser la caméra d'un smartphone ou d'une tablette pour prendre une photo ou enregistrer une vidéo. Nous verrons aussi comment utiliser le microphone de l'appareil pour enregistrer du son et enfin comment lire de l'audio et de la vidéo au sein des applications.

1. Capture

En Codename One, l'emploi de la caméra pour photographier/filmer ou du microphone pour enregistrer de l'audio est géré par une seule classe, relativement simple à utiliser, nommée Capture. Cette classe est constituée uniquement de méthodes statiques, permettant d'effectuer l'une ou l'autre des actions. Ces méthodes existent en versions synchrone et asynchrone.

Encadré : Bon à savoir

Pour prendre une photo ou enregistrer une vidéo, Codename One fera appel à l'application de caméra déjà présente sur le téléphone. Il ne permet pas encore d'intégrer ces fonctionnalités à une interface et de les personnaliser à volonté. De même, pour l'enregistrement audio, il fera appel à l'application d'enregistrement présente par défaut sur l'appareil.

1.1. Exemple : Créer des boutons d'enregistrement

Dans cet exemple, nous allons créer une interface avec trois boutons et un label. L'un des boutons permettra de prendre une photo et de l'afficher dans le label, le deuxième d'enregistrer de l'audio et le dernier de filmer.

```
public void start() {
    final Form f = new Form("Ma superbe application mobile");
    f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
    ❶ Button photo=new Button("Photographier");
    Button audio=new Button("Enregistrer");
    Button video=new Button("Filmer");
    ❷ final Label cadrePhoto=new Label();
    cadrePhoto.getStyle().setAlignment(Component.CENTER);
    photo.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            String cheminPhoto=Capture.capturePhoto();
            ❸ if(cheminPhoto!=null) {
                Image img=null;
                try {
                    img=Image.createImage(cheminPhoto);
                } catch (IOException ex) {
                    Dialog.show("Erreur!", ex.getMessage(), "OK", null);
                }
                cadrePhoto.setIcon(img.scaled(350, 400));
                ❹ f.revalidate();
            }
        }
    });
    audio.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            String cheminAudio=Capture.captureAudio();
            ❺ if(cheminAudio!=null) {
                Dialog.show("Achevé!", "Enregistrement audio terminé", "OK", null);
            }
        }
    });
    video.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            String cheminVideo=Capture.captureVideo();
            ❻ if(cheminVideo!=null) {
                Dialog.show("Achevé!", "Enregistrement vidéo terminé", "OK", null);
            }
        }
    });
    f.add(photo);
    f.add(audio);
    f.add(video);
    f.show();
}
```

❶ Création de trois boutons qui serviront à prendre une photo, à enregistrer de l'audio et de la vidéo.

Création d'un label pour l'affichage de la photo qui sera capturée par le premier
❷ bouton. Nous plaçons ensuite ce label au centre de notre fenêtre avec la méthode
setAlignment() de la classe Style (voir la [Section 4, Styles et transitions](#)).

Appel à la méthode capturePhoto() de la classe Capture pour lancer la caméra et prendre une photo. Une fois la photo prise, elle sera automatiquement enregistrée dans
❸ l'appareil (sous Android par exemple, elle sera placée dans la galerie) et capturePhoto() retournera le chemin complet vers le fichier que nous récupérons dans la variable cheminPhoto.

❹ Création d'une variable de type Image et chargement en mémoire de la photo prise en nous servant du chemin récupéré dans cheminPhoto.

❺ Ajout de l'image chargée au label. Nous redimensionnons l'image avec la méthode scaled() de la classe Image avant de la passer en paramètre à la méthode setIcon().

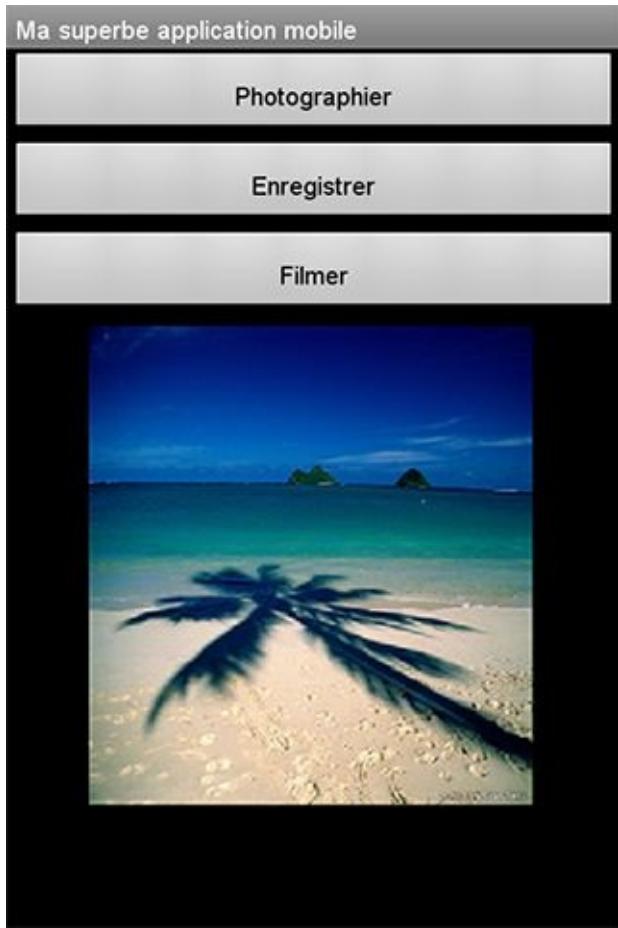
Appel à la méthode revalidate() pour rafraîchir l'interface. Comme nous ajoutons la photo à la fenêtre en cours d'exécution, nous avons besoin de faire appel à cette
❻ méthode pour que l'affichage se passe correctement. C'est une sorte d'actualisation de la fenêtre.

Appel à la méthode captureAudio() pour ouvrir l'application d'enregistrement par
❼ défaut de la plateforme. Tout comme capturePhoto(), elle retourne le chemin complet (dans la variable cheminAudio) vers le fichier audio après enregistrement.

Appel à la méthode captureVideo() pour lancer la caméra et prendre une vidéo. Elle
❽ retourne le chemin complet vers le fichier vidéo récupéré ici dans la variable cheminVideo.

Note > Sur ordinateur, quand vous cliquerez sur ces trois boutons dans le simulateur, une fenêtre s'affichera pour vous demander de choisir un fichier. Ne vous attendez pas alors à prendre des photos ni à faire un enregistrement audio ou vidéo avec le simulateur !

Figure 4.1 : Trois boutons pour photographier, filmer, enregistrer



Comme mentionné plus haut, les méthodes `capturePhoto()`, `captureAudio()` et `captureVideo()` ont leur équivalent asynchrone qui prend en paramètre un `ActionListener`. Si vous optez pour les versions asynchrones de ces méthodes, vous devez alors récupérer différemment le chemin des fichiers créés après la capture. Voici un exemple avec le bouton `Photographier` et qui s'applique aussi aux autres.

```
photo.addActionListener(new ActionListener() {  
  
    public void actionPerformed(ActionEvent evt) {  
        Capture.capturePhoto(new ActionListener() {  
  
            public void actionPerformed(ActionEvent evt) {  
                String cheminPhoto=(String)evt.getSource();  
  
                ❶ if(cheminPhoto!=null) { Image img=null; try { img=Image.createImage(cheminPhoto); } catch (IOException ex) { Dialog.show("Erreur!", ex.getMessage(), "OK", null); } cadrePhoto.setIcon(img.scaled(350, 400)); f.revalidate(); } else { Dialog.show("Annulé", "Photo annulée", "OK", null); } } );  
  
    ❶ Récupération du chemin de la photo.  
});});
```

1.2. Quelques méthodes de Capture

- `boolean hasCamera()` - Vérifie si l'appareil possède une caméra. Retourne `true` si c'est le cas.

2. Lecture

Après avoir capturé de l'audio et de la vidéo, il faut pouvoir les lire. C'est ce que nous allons voir dans cette section. Deux classes s'occupent de la lecture des fichiers audio et vidéo : Media et MediaManager. La première contient toutes les méthodes nécessaires pour effectuer les opérations de lecture (lire, mettre en pause, stopper, ajuster le volume et bien d'autres). La seconde permet de créer la première à partir d'une URL de fichier audio/vidéo ou d'un flux de type InputStream.

2.1. Exemple : Lire un fichier audio ou vidéo

Dans cet exemple, nous allons créer une fenêtre contenant une liste déroulante et un bouton. La liste déroulante propose deux actions : une pour jouer de l'audio et une autre pour jouer de la vidéo. L'élément à jouer dépend du choix effectué dans la liste déroulante.

```
public class TestAudioVideo {  
  
    ❶ private Media m; private InputStream stream; public void init(Object context) { try {  
        Resources theme = Resources.openLayered("/theme");  
        UIManager.getInstance().setThemeProps(  
            theme.getTheme(theme.getThemeResourceNames()[0])); } catch(IOException e){  
        e.printStackTrace(); } } public void start() { final Form f = new Form("Ma superbe  
application mobile"); f.setLayout(new BorderLayout()); final ComboBox choix=new  
ComboBox(); choix.addItem("Jouer de l'audio"); choix.addItem("Jouer de la vidéo");  
Button jouerMedia=new Button("Jouer"); Container c=new Container(new  
BoxLayout(BoxLayout.Y_AXIS)); c.addComponent(choix);  
c.addComponent(jouerMedia); f.addComponent(BorderLayout.NORTH,c);  
jouerMedia.addActionListener(new ActionListener() { public void  
actionPerformed(ActionEvent evt) { int choixAction=choix.getSelectedIndex();  
if(choixAction==0){ try { stream = Display.getInstance()  
.getResourceAsStream(getClass(),"/sintel_song.mp3"); ❷ if(stream!=null) { if(m!=null){  
m.cleanup(); ❸ m=null; } m=MediaManager.createMedia(stream, "audio/mp3"); ❹  
m.play(); ❺ } } catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(),  
"OK", null); } } else if(choixAction==1){ try { stream = Display.getInstance()  
.getResourceAsStream(getClass(),"/sintel_trailer.mp4"); if(stream!=null) { if(m!=null){  
m.cleanup(); m=null; } m=MediaManager.createMedia(stream, "video/mp4");  
f.addComponent(BorderLayout.CENTER,m.getVideoComponent()); ❻ f.revalidate();  
m.play(); } } catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK",  
null); } } ); f.show(); } public void stop() {} public void destroy() {} }
```

❶ Déclaration d'un objet Media qui servira à jouer l'élément audio ou vidéo. Déclaration ❷ d'un élément InputStream pour le chargement du fichier à lire.

- ② Création d'un flux InputStream et chargement du fichier audio `sintel_song.mp3` placé dans le dossier SRC du projet. Ce fichier sera intégré à l'exécutable de l'application pendant la compilation.

La méthode `cleanup()` utilisée ici permet de stopper la lecture en cours et de libérer les ressources allouées par l'objet `Media`. Nous appelons cette méthode parce que dans notre exemple la même variable `Media` est utilisée pour lire à la fois la vidéo et l'audio.

- ③ Pour cela, nous vérifions d'abord si cet objet est null ou non. S'il ne l'est pas, alors une lecture est déjà en cours donc nous arrêtons cette lecture avec `cleanup()`, nous l'initialisons à null, puis nous l'utilisons à nouveau pour une nouvelle lecture.

Création de l'objet `Media` avec la méthode `createMedia()` de la classe `MediaManager`.

- ④ Cette méthode a d'autres variantes et celle que nous utilisons ici prend en premier paramètre l'objet `InputStream` qui contient le fichier à lire et en deuxième paramètre le type MIME du fichier. Ici, il s'agit d'un fichier MP3.

- ⑤ Appel à la méthode `play()` pour lancer la lecture.

Ici, nous lisons de la vidéo donc nous allons ajouter un composant visuel à notre

- ⑥ fenêtre. Pour récupérer ce composant, nous utilisons la méthode `getVideoComponent()` de la classe `Media`.

Attention > N'oubliez pas le slash(“/”) placé devant les noms des fichiers audio et vidéo.

Figure 4.2 : Lecture d'un fichier vidéo et audio (vidéo)



2.2. Lire un fichier audio ou vidéo depuis Internet

Au lieu de charger le fichier audio ou vidéo en local, vous pouvez aussi le charger depuis une adresse internet. Pour cela, utilisez l'une des variantes de la méthode `createMedia()`. Par exemple :

```
Media  
m=MediaManager.createMedia("http://www.votresite.com/sintel_trailer.mp4",  
true);
```

Le premier paramètre est l'adresse du fichier et le deuxième un booléen qui permet de préciser si le fichier à lire est une vidéo. Si c'est le cas, alors mettez la valeur `true`.

2.3. Lire un fichier en boucle

Si vous avez besoin de jouer un fichier audio ou vidéo en boucle, vous pouvez le faire, mais vous ne trouverez aucune méthode dédiée dans la classe `Media`. Pour cela, vous devrez utiliser une autre variante de la méthode `createMedia()`, qui prend un troisième paramètre de type `Runnable`. Voici comment procéder :

```
private Runnable onComplete;  
  
//... autres codes ici...  
  
try {  
    stream = Display.getInstance().getResourceAsStream(getClass(),  
"/sintel_song.mp3");  
    if(stream!=null) {  
        if(m!=null){  
            m.cleanup();  
            m=null;  
        }  
    }  
  
    onComplete=new Runnable() {  
  
❶ public void run() { try {  
        stream=Display.getInstance().getResourceAsStream(getClass(), "/sintel_song.mp3");  
        m=MediaManager.createMedia(stream, "audio/mp3",onComplete); m.play(); } catch  
(IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK", null); } } };  
    m=MediaManager.createMedia(stream, "audio/mp3", onComplete);❷ m.play(); } }  
catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK", null); } //...  
autres codes ici...
```

Par défaut, un objet `Media` s'autodétruit à la fin de la lecture d'un fichier. Pour y remédier et continuer à lire un fichier audio, nous utilisons la variante de `createMedia()` qui prend comme troisième paramètre un objet de type `Runnable`, lequel sera appelé chaque fois que la lecture du fichier sera terminée ❷. Le rôle de `Runnable` étant de créer un `thread`, nous répétons l'opération de lecture dans ce `thread` ❶. C'est ainsi que l'objet `onCompletion` (de type `Runnable`) sera appelé à la fin de chaque lecture pour répéter la même opération de lecture. Voilà ! Vous avez maintenant de quoi lire vos fichiers vidéo et audio en boucle.

2.4. Quelques méthodes de Media

- `void pause()` – Met en pause la lecture en cours.
- `void setTime(int time)` – Définit la position où la lecture doit débuter. La valeur du paramètre `time` doit être exprimée en millisecondes.
- `void setVolume(int vol)` – Définit en pourcentage le niveau du volume.
- `void setFullScreen(boolean full)` - Indique au média de s'afficher en mode plein écran s'il s'agit d'une vidéo.
- `boolean isPlaying()` – Vérifie s'il y a une lecture en cours. Si c'est le cas, elle retourne la valeur `true`.
- `int getVolume()` – Retourne le niveau du volume.
- `int getDuration()` – Retourne la durée totale du fichier à lire. Si cette valeur n'est pas connue alors elle retourne la valeur `-1`.

3. Le composant MediaPlayer

MediaPlayer est un composant qui fournit un lecteur avec des boutons lecture, pause et avance rapide. Il permet de contrôler la lecture d'un média sans avoir à créer soi-même ces boutons. MediaPlayer peut aussi bien lire un fichier en local que depuis une adresse URL.

3.1. Exemple : Interface avec lecteur multimédia intégré

```
Form f = new Form("Ma superbe application mobile");
f.setLayout(new BorderLayout());

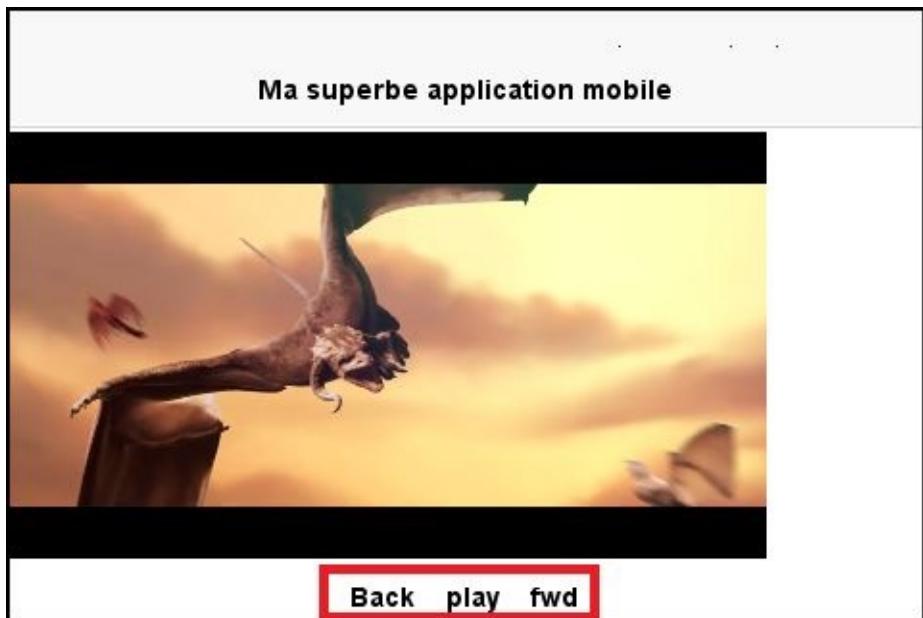
MediaPlayer player=null;
InputStream stream;

try {
    stream = Display.getInstance().getResourceAsStream(getClass(),
"/sintel_trailer.mp4");
    if(stream!=null) {
        if(m!=null){
            m.cleanup();
            m=null;
        }
        m=MediaManager.createMedia(stream, "video/mp4");
        player=new MediaPlayer(m);
    }
} catch (IOException ex) { Dialog.show("Erreur", ex.getMessage(), "OK", null); }
f.addComponent(BorderLayout.CENTER, player); ② f.show();
```

① Création de l'objet MediaPlayer avec en paramètre le média à lire.

② Ajout du lecteur MediaPlayer à la fenêtre Form.

Figure 4.3 : Un lecteur multimédia



Note > Par défaut, les options de lecture d'un MediaPlayer s'affichent en anglais (voir zone encadrée de la [Figure 4.3](#)) mais nous pouvons le modifier en traduisant ces mots en français. Nous parlerons de la [traduction](#) d'une application au Chapitre [Codename One Designer](#).

Pour lire de la vidéo depuis une adresse internet, utilisez la méthode `setDataSource()` de la classe MediaPlayer pour charger le fichier. Une variante de cette méthode prend aussi en paramètre un InputStream, le type MIME et un Runnable comme vu à la [Section 2.3, Lire un fichier en boucle](#) :

```
mp.setDataSource("http://www.votresite.com/sintel_trailer.mp4");
```

3.2. Quelques méthodes de MediaPlayer

- `void setPlayIcon(Image icon)` – Attribue une icône au bouton Play qui permet de jouer la vidéo.
- `void setPauseIcon(Image icon)` – Attribue une icône au bouton Pause qui permet de mettre la vidéo en pause.
- `void setFwdIcon(Image icon)` – Attribue une icône au bouton Fwd qui permet de faire une lecture rapide de la vidéo.
- `void setBackIcon(Image icon)` - Attribue une icône au bouton Back qui permet de revenir en arrière dans la vidéo.
- `void setDataSource(InputStream stream, String typeMime, Runnable r)` – Définit la source de données à lire. C'est une variante de [celle que nous avons vue](#). Elle prend les mêmes paramètres que la [variante de createMedia\(\)](#) qui permettait de lire un fichier en boucle : le flux InputStream qui chargera le fichier, le type MIME du fichier à lire et un objet de type Runnable.

4. Accès à la galerie d'images et de vidéos

Il est possible d'accéder à la galerie de l'appareil pour choisir une image ou une vidéo à l'aide de la méthode `openGallery()` du singleton `Display`, qui prend en paramètre un listener. Son utilisation est simple et la voici :

```
Display.getInstance().openGallery(new ActionListener() {  
    public void actionPerformed(ActionEvent evt) {  
        if(evt!=null){  
            String cheminImage Ou VideoChoisie=(String)evt.getSource();  
  
① // Actions à effectuer ici avec le chemin de l'image ou de la vidéo } } },  
Display.GALLERY_ALL);
```

① Récupération du chemin complet vers l'image ou la vidéo choisie dans la galerie.

Le deuxième paramètre de `openGallery()` est une constante qui permet de préciser si la galerie doit afficher uniquement les images, les vidéos ou les deux. Dans notre exemple, nous avons choisi d'afficher les deux avec la valeur `Display.GALLERY_ALL`. Pour les images uniquement, utilisez `Display.GALLERY_IMAGE` et pour les vidéos uniquement, utilisez `Display.GALLERY_VIDEO`.

Note > Sous iOS et Android, `openGallery()` ouvrira la galerie de l'appareil. Sous les autres plateformes, elle ouvrira un explorateur de fichiers qui permettra de parcourir un espace de stockage et de sélectionner l'image ou la vidéo de son choix.

Réseau, Internet et services web

Dans un monde de plus en plus connecté, faire communiquer une application avec Internet devient normal et très fréquent. C'est ce que nous allons apprendre à faire dans ce chapitre. En pratique, nous allons apprendre à télécharger des fichiers, à uploader des fichiers en ligne, à communiquer avec un service web quelconque et nous terminerons en apprenant comment échanger des données avec une base de données distante. À l'instant où ces lignes sont écrites, Codename One ne supporte que les connexions HTTP et HTTPS.

Attention > *Il est conseillé d'avoir des notions sur le fonctionnement du protocole HTTP pour tirer profit de ce chapitre.*

Note > *Dans certaines sections de ce chapitre, vous aurez besoin d'utiliser un langage de développement web pour écrire les codes qui devront s'exécuter côté serveur. Le langage choisi ici est le PHP et pour l'utiliser vous aurez besoin d'installer un serveur web et une base de données. Pour cela, vous pourrez télécharger et installer l'un des logiciels suivants : [EasyPHP](#), [Wamp](#), [Xampp](#). Si vous êtes plutôt fan d'un autre langage web alors vous savez les outils à installer.*

1. Gestion de la connexion

NetworkManager et ConnectionRequest sont les deux classes principales de manipulation du réseau en Codename One. La première permet de gérer les requêtes de connexion et la seconde de créer ces requêtes. La plupart du temps, un traitement utilisant le réseau est placé dans un thread parallèle à celui de l'interface graphique. Ainsi, l'un des problèmes les plus fréquents en programmation réseau est la gestion de ces threads multiples. En Codename One, la classe NetworkManager crée et gère automatiquement les threads en parallèle au thread de l'[EDT](#). Elle nous épargne ainsi certaines contraintes liées à la gestion manuelle de ces threads consacrés au traitement du réseau.

La création d'une requête est gérée par la classe ConnectionRequest et chaque requête peut être envoyée de manière synchrone ou asynchrone. Les erreurs de traitement sont gérées par défaut, mais il est possible de les gérer manuellement en utilisant des listeners. Une fois des requêtes de connexion créées avec ConnectionRequest, il faut les ajouter à une file d'attente avec NetworkManager, qui se chargera non seulement de les exécuter, mais aussi de créer pour chacune des requêtes un thread. L'envoi de la requête, la destruction du thread réseau, etc. sont aussi gérés par NetworkManager.

Dans le cas de la communication avec un web qui peut renvoyer des données de type XML ou JSON, Codename One fournit des classes (XMLParser et JSONParser) pour la manipulation de ces formats.

La création d'une requête basique pourrait ressembler à ceci :

```
ConnectionRequest requete=new ConnectionRequest();
❶ requete.setUrl("ADRESSE"); ❷ requete.setPost(false); ❸
requete.setContentType("TYPE MIME DU CONTENU"); ❹
requete.addArgument("NOM d'UN ARGUMENT", "VALEUR DE L'ARGUMENT"); ❺
requete.setPriority(ConnectionRequest.PRIORITY_NORMAL); ❻
requete.addResponseListener(new ActionListener() { ❻ public void
actionPerformed(ActionEvent evt) { //Récupération et traitement des données ici } });
NetworkManager.getInstance().addToQueue(requete); ❻
```

❶ Création de la requête avec une instanciation de ConnectionRequest.

❷ La méthode setUrl() permet de définir l'adresse qui servira à la requête.

La valeur false de la méthode setPost() précise que nous voulons récupérer des ❸ données et non en envoyer (valeur true). Il s'agit du type de méthode HTTP à utiliser (GET ou POST).

❹ setContentType() permet de définir le type MIME de la donnée.

❺ addArgument() permet d'ajouter des arguments à la requête. En premier paramètre le nom de l'argument et en deuxième paramètre sa valeur.

La définition de la priorité de la requête dans la file d'attente se fait avec `setPriority()`. Ici, nous précisons une priorité normale avec la valeur `PRIORITY_NORMAL`. Les autres valeurs possibles sont : `PRIORITY_HIGH` (priorité haute), `PRIORITY_LOW` (priorité basse), `PRIORITY_CRITICAL` (priorité critique). Cette dernière priorité est supérieure à `PRIORITY_HIGH`, `PRIORITY_REDUNDANT` (priorité redondante). Elle permet d'écartez de la liste d'attente les éléments redondants mis en pause. Par défaut, la priorité normale est utilisée quand aucune priorité n'est définie.

⑥ `addResponseListener()` nous permet de connecter la requête à un `ActionListener` pour la récupération et le traitement de la réponse renvoyée par la requête. Il existe un autre moyen pour faire la même action de manière plus détaillée que nous verrons à l'[Exemple 5.1](#).

⑦ La méthode `addToQueue()` permet d'ajouter la requête créée à une file d'attente d'exécution créée par `NetworkManager`.

1.1. Téléchargement de données

Dans cette section, nous allons voir plusieurs manières de télécharger des données sur Internet : à l'aide des classes [NetworkManager](#) et [ConnectionRequest](#), avec les classes [ImageDownloadService](#) et [URLImage](#) et enfin en recourant à certaines [méthodes de la classe Util](#).

Avec NetworkManager et ConnectionRequest

Nous allons maintenant passer à des exemples concrets de création de requêtes et de récupération des réponses. Le premier exemple nous permettra de télécharger en ligne un fichier MP3, d'enregistrer ce fichier sur le téléphone et de le jouer ensuite. Nous allons aussi apprendre une autre façon de récupérer des données renvoyées par la requête sans utiliser la méthode `addResponseListener()`.

Exemple 5.1 : Téléchargement d'un fichier MP3 et lecture du fichier téléchargé

```
public class TestCN1Reseau implements ActionListener {  
  
    private Form current;  
    private Button telechargerMusique;  
  
    public void init(Object context) {  
        try {  
            Resources theme=Resources.openLayered("/theme");  
            UIManager.getInstance().setThemeProps(  
                theme.getTheme(theme.getThemeResourceNames()[0]));  
        } catch(IOException e){  
            e.printStackTrace();  
        }  
    }  
}
```

```

        }

    public void start() {
        if(current!=null){
            current.show();
            return;
        }

        Form f=new Form("Test de téléchargement");
        f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
    }
}

```

❶ telechargerMusique=**new** Button("Télécharger la musique");
f.addComponent(telechargerMusique); telechargerMusique.addActionListener(this);
f.show(); } **public void** stop() { current=Display.getInstance().getCurrent(); } **public void**
destroy() { } **public void** actionPerformed(ActionEvent evt) { Object obj=evt.getSource();
❷ if(obj==telechargerMusique){ **❸** ConnectionRequest requete=**new**
ConnectionRequest(){ **❹** **❺** FileSystemStorage fs=FileSystemStorage.getInstance();
String chemin=fs.getAppHomePath(); **@Override protected void** postResponse() { **❻**
if(Dialog.show("Terminé", "Téléchargement terminé", "OK", null)){ InputStream
in=null; **❷ try**{ in=fs.openInputStream(chemin+"Expedition.mp3"); Media
m=MediaManager.createMedia(in, "audio/mp3"); m.play(); } **catch**(IOException ex){
Dialog.show("Erreur", ex.getMessage(), "OK", null); } **finally** { Util.cleanup(in); } } }
@Override protected void readResponse(InputStream input) **throws** IOException { **❸**
OutputStream out=null; **❹ try** { out=fs.openOutputStream(chemin+"Expedition.mp3");
Util.copy(input, out); } **catch**(IOException ex) { Dialog.show("Erreur", "Erreur
inattendue: "+ex.getMessage(), "OK", null); } **finally** { Util.cleanup(out); } } }; **❽**
requete.setUrl("http://localhost/Expedition.mp3"); requete.setPost(false);
requete.setContentType("audio/mp3"); **❻**
NetworkManager.getInstance().addToQueue(requete); **❾** InfiniteProgress progress=**new**
InfiniteProgress(); Dialog d=progress.showInfiniteBlocking();
requete.setDisposeOnCompletion(d); **❿ // Progress p=new Progress("Patientez SVP",**
requete); // p.setDisposeOnCompletion(true); // p.show(); } } }

❶ Création d'un bouton permettant de télécharger un fichier MP3. Ajout du bouton à la fenêtre Form et connexion du bouton au ActionListener.

❷ Récupération de l'objet ayant émis un signal.

❸ Si l'objet récupéré qui a émis le signal est le bouton de téléchargement de musique alors, exécuter les actions qui suivent.

❹ Création de l'objet ConnectionRequest qui servira à créer une requête.

Création d'une instance de FileSystemStorage qui servira à stocker la musique téléchargée sur le téléphone. La création de cet objet est suivie de la récupération du

chemin du dossier principal du stockage des données de l'application.

postResponse() est une méthode héritée de ConnectionRequest que nous avons redéfinie et implémentée. Cette méthode est automatiquement appelée à la fin de

- ⑥ l'exécution de chaque requête. Son invocation se fait dans l'**EDT** par le thread du réseau et permet de mettre à jour l'interface graphique de l'application. Dans cet exemple, nous affichons une boîte de dialogue qui informe par un message de la fin du téléchargement.

La fermeture de la boîte de dialogue précédemment affichée permettra la création d'un objet Media dans le but de jouer la musique téléchargée. La création de cet objet Media

- ⑦ est précédée de la création d'un flux de lecture InputStream avec le chemin vers le fichier MP3 téléchargé.

readResponse() est aussi une méthode de ConnectionRequest que nous avons

- ⑧ redéfinie et implémentée. Elle renvoie les données récupérées par la requête et ces données sont disponibles à travers son paramètre nommé `input`.

Création d'un flux d'écriture de données OutputStream avec le chemin où le fichier téléchargé sera sauvegardé dans le téléphone. La création de ce flux est suivie de la copie des données binaires à l'intérieur du paramètre `input` (de la méthode

readResponse()) vers le flux de sortie OutputStream créé. Cette copie se fait ici avec la méthode `copy()` de la classe Util et permettra l'écriture des données téléchargées en ligne à l'intérieur du fichier créé dans le téléphone.

Note > Si vous vous demandez pourquoi une autre méthode (`cleanup()` de la classe Util) a été utilisée pour la fermeture du flux au lieu de la méthode `close()` du OutputStream, voici la réponse. Vous pouvez utiliser la méthode `close()` du flux pour le fermer. La seule différence avec `cleanup()` est que cette dernière ne déclenchera pas d'exception même si la valeur du flux est nulle.

Paramétrage de la requête créée avec l'ajout de l'URL du fichier à télécharger avec

- ⑩ `setUrl()` suivi de l'ajout du type de la requête. Ici, une requête de type GET (avec le paramètre de la méthode `setPost()` passé à `false`) pour récupérer des données. Enfin, l'ajout du type MIME du fichier à télécharger.

Création d'une instance de NetworkManager et ajout de la requête créée à la file

- ⑪ d'attente d'exécution avec la méthode `addToQueue()`. `addToQueue()` étant asynchrone, il existe aussi sa version bloquante nommée `addToQueueAndWait()`.

InfiniteProgress permet de créer une fenêtre transparente avec l'image de roue qui tourne indéfiniment. Nous l'utilisons ici pour représenter l'exécution en cours d'opération. Il est conseillé de l'utiliser pour des opérations longues afin que

l'utilisateur n'ait pas l'impression que rien ne se passe. Après la création de l'objet InfiniteProgress, nous appelons la méthode `showInfiniteBlocking()` pour afficher la

- ⑫ fenêtre transparente (avec la petite roue qui tourne au milieu) qui couvrira l'écran de

l'application. Cette méthode retourne un objet de type Dialog que nous passons en paramètre à la méthode `setDisposeOnCompletion()` de la requête `ConnectionRequest`. `setDisposeOnCompletion()` permet de fermer la fenêtre créée par `InfiniteProgress` une fois l'exécution de la requête terminée.

Le code placé à cet endroit est placé en commentaire et vous pouvez l'utiliser pour remplacer celui du point ⑫. La classe `Progress` crée une boîte de dialogue qui affichera la valeur de la progression de l'opération (du téléchargement dans notre cas) en cours. Cette boîte de dialogue contient un bouton pour annuler l'opération. Dans le cas de cet exemple, nous utilisons le constructeur qui prend en premier paramètre un texte qui sera affiché à l'utilisateur (ici, nous avons placé le texte `Patientez SVP`) et en deuxième paramètre l'objet `requete`. Ensuite nous passons à `true` le paramètre de la méthode `setDisposeOnCompletion()` (de la classe `Progress`) qui joue le même rôle que celle de la classe `ConnectionRequest` utilisée au point ⑫. Pour finir, nous affichons la boîte de dialogue créée par `Progress` avec la méthode `show()`.

Figure 5.1 : Téléchargement en cours (avec `InfiniteProgress`)

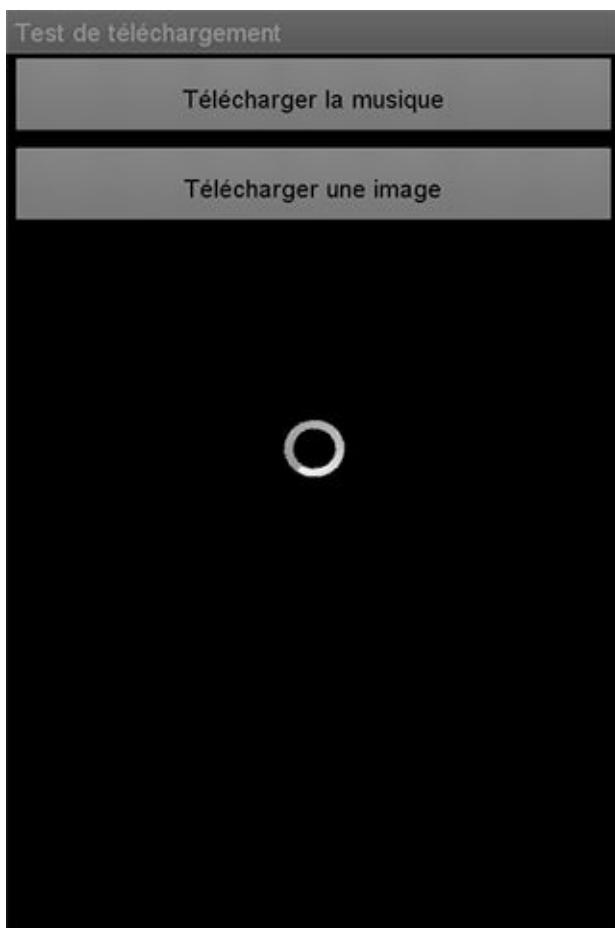


Figure 5.2 : Téléchargement en cours (avec `Progress`)



La méthode de traitement des données de la requête utilisée dans l'exemple se résume à ceci :

```
ConnectionRequest requete=new ConnectionRequest() {  
  
protected void readResponse(InputStream input) {  
// Lecture de la réponse de la requête à partir du InputStream présent en paramètre  
}  
  
protected void postResponse() {  
// Action à effectuer à la fin de l'exécution de la requête. Cette action se passe à l'intérieur de l'EDT et peut servir par exemple à mettre à jour l'interface graphique.  
}  
};
```

Note > En ce qui concerne la requête, ConnectionRequest a d'autres méthodes dont vous pouvez redéfinir les comportements comme nous l'avons fait avec `postResponse()` et `readResponse()`. Il vous suffit de consulter la [Javadoc de cette classe](#) pour les découvrir.

Figure 5.3 : Confirmation de fin de téléchargement venant de `postResponse()`



Au début de ce chapitre, nous avons vu qu'on pouvait également utiliser un listener pour récupérer les données renvoyées par la requête. En voici un exemple :

```
ConnectionRequest requete=new ConnectionRequest() ;  
//...  
requete.addResponseListener(new ActionListener() {  
  
    public void actionPerformed(ActionEvent evt) {  
        NetworkEvent ne=(NetworkEvent)evt;  
        byte[] data=(byte[])ne.getMetaData(); //Récupération des données sous  
        // forme de byte. À convertir dans le type que vous voulez.  
        // Traitement des données ici  
    }  
}
```

À la différence de cette méthode, celle que nous avons utilisée à l'[Exemple 5.1](#) permet d'avoir plus de contrôle parce qu'elle nous permet de redéfinir le comportement des méthodes de ConnectionRequest. Je la conseillerais donc davantage, mais il vous revient de savoir ce que voulez faire et la manière dont vous voulez le faire.

Avec ImageDownloadService

ImageDownloadService est une classe fille de ConnectionRequest. Elle permet de simplifier le téléchargement d'une image sans écrire trop de code avec NetworkManager et ConnectionRequest. À la fin du téléchargement, une image est créée et associée

directement à un composant de type Label ou List. Avant d'être associée à un Label ou à une List, l'image sera placée en cache. Une image téléchargée avec ImageDownloadService peut être placée dans un Storage ou dans un fichier (si vous avez l'intention d'utiliser le fichier de l'image même).

L'extrait de code suivant permet de télécharger une image et de l'afficher sur la fenêtre en cours après un clic sur un bouton. Nous nous contenterons de l'intégrer à celui de l'[Exemple 5.1](#). Déclarez un second bouton et un label sous forme de variables d'instance et nommez-les respectivement telechargerImage et image. Déclarez aussi l'objet Form en tant que variable d'instance. Ajoutez ensuite le second bouton et le label au Form puis connectez le bouton à l'ActionListener représenté ici par notre classe TestCN1Reseau.

Exemple 5.2 : Téléchargement d'une image avec ImageDownloadService

```
public class TestCN1Reseau implements ActionListener {  
  
    //...  
    private Button telechargerImage;  
    private Label image;  
    private Form f  
  
    //...  
    public void start() {  
        //...  
        f=new Form("Test de téléchargement");  
        // ...  
        telechargerImage=new Button("Télécharger une image");  
        image=new Label();  
  
        //...  
        f.addComponent(telechargerImage);  
        f.addComponent(image);  
        // ...  
        telechargerImage.addActionListener(this);  
        //...  
    }  
  
    //...  
}
```

Dans la méthode actionPerformed(), où les événements sont traités, ajoutons le code suivant :

```
if(obj==telechargerMusique){  
    //...  
} else if(obj==telechargerImage){  
    ImageDownloadService.createImageToStorage("http://localhost/photo.jpg",  
    image, "photo", null);  
  
❶ f.revalidate(); }
```

La méthode createImageToStorage() utilisée ici permet de télécharger une image, de l'ajouter à un Storage et de l'associer à un Label pour l'affichage. Le premier paramètre est l'adresse de l'image à télécharger, le deuxième est le Label qui va

accueillir l'image sur l'interface graphique une fois qu'elle est téléchargée, le troisième ❶ est le nom donné au Storage qui sera créé pour stocker l'image et le dernier paramètre est la taille ou dimension à donner à l'image avant son affichage. Si vous voulez qu'elle s'affiche à sa taille d'origine alors mettez `null` comme valeur de ce paramètre. `createImageToStorage()` est une méthode surchargée, donc vous en trouverez d'autres variantes dans la [Javadoc](#).

Si vous préférez stocker l'image téléchargée dans un fichier plutôt que dans un Storage, recourez à la méthode `createImageToFileSystem()` qui peut s'utiliser de la manière suivante :

```
String chemin= FileSystemStorage.getInstance().getAppHomePath();
ImageDownloadService.createImageToFileSystem("http://localhost/photo.jpg",
this, chemin+"photo.jpg");
```

En premier paramètre de `createImageToFileSystem()`, nous avons l'URL de l'image. En deuxième paramètre, un objet de type `ActionListener` qui peut effectuer des opérations une fois que la donnée téléchargée est disponible. Ici nous utilisons le mot clé `this` pour désigner notre classe, mais nous n'effectuons aucune action à ce niveau. Le dernier paramètre est le nom du fichier de destination sur l'appareil. Il doit contenir le chemin complet du fichier qui sera créé. `createImageToFileSystem()` est aussi surchargée. Voir la [Javadoc](#) pour plus de variantes de la même méthode.

Avec URLImage

`URLImage` a le même rôle qu'`ImageDownloadService`. Elle apporte cependant d'autres fonctionnalités intéressantes comme le fait d'utiliser une image temporaire qui sera automatiquement remplacée par l'image téléchargée, d'adapter la taille de l'image téléchargée à la taille de l'image temporaire, de mettre l'image téléchargée en cache pour pouvoir la réutiliser en cas de besoin sans avoir à la télécharger de nouveau, de faire une mise à jour en cas de modification de l'image téléchargée. Tout comme `ImageDownloadService`, elle permet de sauvegarder l'image téléchargée dans un Storage ou dans un fichier classique. Voici un exemple d'utilisation.

Exemple 5.3 : Téléchargement d'une image avec URLImage

```
//...
private EncodedImage ei;

//...
public void start() {
    final Form f = new Form("URLImage");
    f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

    final Label image=new Label();
    image.getStyle().setAlignment(Component.CENTER);
```

```

try {
    ei = EncodedImage.create("/chambre.jpg");
} catch (IOException ex) {
    Dialog.show("Erreur", ex.getMessage(), "OK", null);
}

Button b=new Button("Télécharger l'image");
b.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        Image img=URLImage.createToStorage(ei, "MaPhoto",
"http://localhost/burj-al-arab.jpg", URLImage.RESIZE_SCALE);

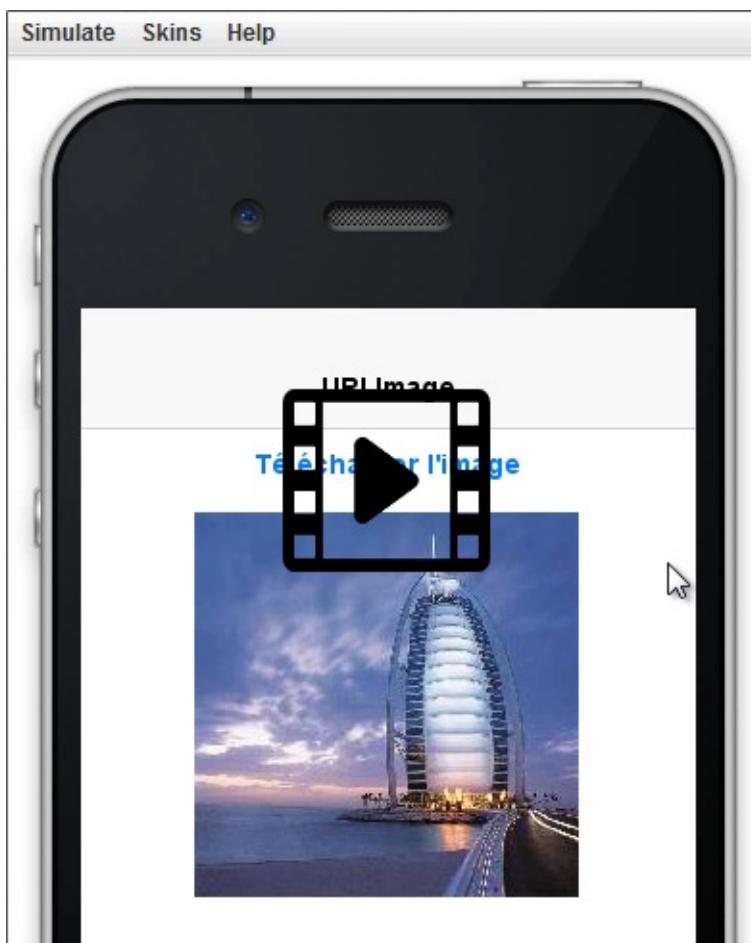
❶ image.setIcon(img); f.revalidate(); } }); f.addComponent(b); f.addComponent(image);
f.show(); }

```

La méthode `createToStorage()` télécharge une image d'une URL vers un Storage. Son premier paramètre représente une image temporaire qui sera affichée pendant que le téléchargement de la nouvelle image sera en cours. Le deuxième paramètre est le nom du Storage où l'image sera stockée. Le troisième paramètre est l'URL de l'image à télécharger et le dernier paramètre est une valeur qui permet d'adapter la taille de l'image à la taille de l'emplacement où elle sera affichée. Les valeurs possibles de ce paramètre sont `RESIZE_SCALE`, `RESIZE_SCALE_TO_FILL` et `RESIZE_FAIL`.

❶

Figure 5.4 : URLImage en action (vidéo)



Pour stocker l'image téléchargée dans un fichier classique plutôt que dans un Storage, utilisez la méthode `createToFileSystem()` qui prend les mêmes paramètres que `createToStorage()` sauf en deuxième paramètre où il faut mettre le nom du fichier à stocker. Ce nom doit être précédé du chemin complet de l'emplacement de stockage.

Avec certaines méthodes de la classe Util

La classe Util, qui ne contient que des méthodes statiques, fournit quelques méthodes qui vous simplifieront le téléchargement de toutes sortes de fichiers sans avoir recours manuellement aux classes NetworkManager et ConnectionRequest. Utilisez-les, elles vous feront gagner du temps sur des opérations simples de téléchargement.

- `void downloadUrlToFile(String url, String filename, boolean showProgress)` – Télécharge un fichier d'une URL vers un fichier créé en local sur le téléphone. Le premier paramètre est l'adresse du fichier à télécharger ; le deuxième, le nom du fichier de destination (incluant le chemin complet) et le dernier paramètre un booléen qui permet d'activer ou de désactiver l'affichage d'une fenêtre de progression. Il s'agit en fait d'un composant [InfiniteProgress](#).

Cette méthode est bloquante et ne retourne qu'à la fin de l'opération. Sa version asynchrone (non bloquante) est `downloadUrlToFileSystemInBackground()`. Elle existe en deux versions et l'une d'elles prend les mêmes paramètres que `downloadUrlToFile()`.

- `void downloadUrlToStorage(String url, String filename, boolean showProgress)` – Elle est semblable à `downloadUrlToFile()` à la différence près que le fichier téléchargé n'est pas stocké dans un fichier classique, mais dans un Storage. Les trois paramètres sont les mêmes que ceux de `downloadUrlToFile()`. Cette méthode aussi est bloquante. L'équivalent asynchrone est `downloadUrlToStorageInBackground()`. Elle existe également en deux versions et l'une d'elles prend les mêmes paramètres que `downloadUrlToStorage()`.

En pratique, l'utilisation de l'une de ces méthodes ressemblerait à ceci :

```
String chemin=FileSystemStorage.getInstance().getAppHomePath();
Util.downloadUrlToFile("http://localhost/clip.mp4", chemin+"clip.mp4",
true);
```

1.2. Envoi de données (upload) avec MultipartRequest

De la même façon qu'on peut télécharger des données ou des fichiers, on peut aussi envoyer des fichiers en ligne vers un serveur distant. S'il est tout à fait possible de le faire

en partant de zéro à l'aide des classes NetworkManager et ConnectionRequest et de la méthode d'envoi POST, sachez que Codename One fournit une classe adaptée à ce besoin, surtout pour l'envoi de gros fichiers en ligne : la classe MultipartRequest. Elle hérite de ConnectionRequest et ne présente pas les limitations des requêtes ordinaires quand il s'agit d'envoyer de gros blocs de données.

Dans l'[Exemple 5.4](#), nous allons permettre à l'utilisateur de choisir un fichier MP3 quelconque dans son téléphone et de l'envoyer vers un serveur en ligne. *Il vous revient d'écrire avec le langage de développement web de votre choix, le code de récupération du fichier sur le serveur. Dans le cadre de cet exemple, nous allons récupérer le fichier uploadé avec du code PHP.*

Note > Les outils dont vous avez besoin pour exécuter du code PHP sur votre ordinateur ont été mentionnés dans l'introduction de ce chapitre.

Exemple 5.4 : Upload d'un fichier MP3 vers un serveur en ligne

```
final String URL_DESTINATION="http://localhost/upload.php";
Form f=new Form("Choisissez un fichier");
f.setLayout(new BorderLayout());

Image dossier_ouvert=null;
Image dossier_ferme=null;
try{
    dossier_ouvert=Image.createImage("/dossier_ouvert.png");
    dossier_ferme=Image.createImage("/dossier_ferme.png");

} catch(IOException ex){
    Dialog.show("Erreur de chargement", ex.getMessage(), "Ok", null);
}

FileTree.setFolderIcon(dossier_ferme);

❶ FileTree.setFolderOpenIcon(dossier_ouvert); ❷ FileTree ft=new FileTree();
❸ ft.addLeafListener(new ActionListener() { ❹ public void actionPerformed(ActionEvent evt) {
    String fichierChoisi=(String)evt.getSource(); ❺
    if(!fichierChoisi.toLowerCase().endsWith(".mp3")){
        Dialog.show("Fichier invalide",
        "Le fichier choisi n'est pas un fichier MP3", "Ok", null);
        return;
    }
    if(Dialog.show("Voulez-vous envoyer ce fichier?", fichierChoisi, "Oui", "Non")){
        ❻ InfiniteProgress iprogress=new InfiniteProgress();
        Dialog progress=iprogress.showInifiniteBlocking(); ❽ MultipartRequest requete=new
        MultipartRequest(); requete.setUrl(URL_DESTINATION); requete.setPost(true); try {
        requete.addData("contenu", fichierChoisi, "audio/mp3"); ❾
        requete.setDisposeOnCompletion(progress);
        NetworkManager.getInstance().addToQueue(requete); ❾ } catch (IOException ex) {
        Dialog.show("Erreur", ex.getMessage(), "Ok", null); } } });
    f.addComponent(BorderLayout.CENTER, ft); f.show();
```

❶ La méthode `setFolderIcon()` de la classe `FileTree` (voir le point ❸) permet de définir l'icône des dossiers non ouverts (contenus non déroulés).

② La méthode `setFolderOpenIcon()` de `FileTree` (point ③) permet de définir l'icône des dossiers ouverts (contenus déroulés).

Création d'un `FileTree`. Ce composant permet de créer un arbre hiérarchique du ③ système de fichiers du téléphone. Cela permet ainsi de parcourir les fichiers des différents espaces de stockage.

Ajout d'un listener au `FileTree` à travers `addLeafListener()`. Cette méthode déclenche ④ un listener une fois qu'un fichier a été choisi dans la liste des fichiers affichée par le `FileTree`.

⑤ Récupération du chemin complet et nom du fichier MP3 sélectionné pour l'envoi vers le serveur.

Une fois qu'un fichier est sélectionné, une demande de confirmation est affichée à ⑥ l'utilisateur pour savoir s'il veut vraiment envoyer le fichier en ligne. S'il répond par l'affirmatif, alors la création et l'exécution de la requête sont déclenchées.

Déclaration de l'objet `MultipartRequest` qui servira à créer la requête suivie de l'ajout ⑦ de l'adresse vers laquelle le fichier sera envoyé et spécification de la méthode d'envoi. En l'occurrence, il s'agit de la méthode `POST` définie par le fait que `setPost()` prenne `true` en paramètre.

Ajout du fichier à envoyer avec la méthode `addData()`. En premier paramètre de cette ⑧ méthode, nous avons le nom de la donnée à envoyer. En deuxième paramètre, le chemin complet vers le fichier à uploader puis en troisième paramètre le type MIME du fichier.

⑨ Ajout de la requête à la file d'attente d'exécution. Lance aussi l'exécution de la requête.

Figure 5.5 : Choix du fichier à uploader

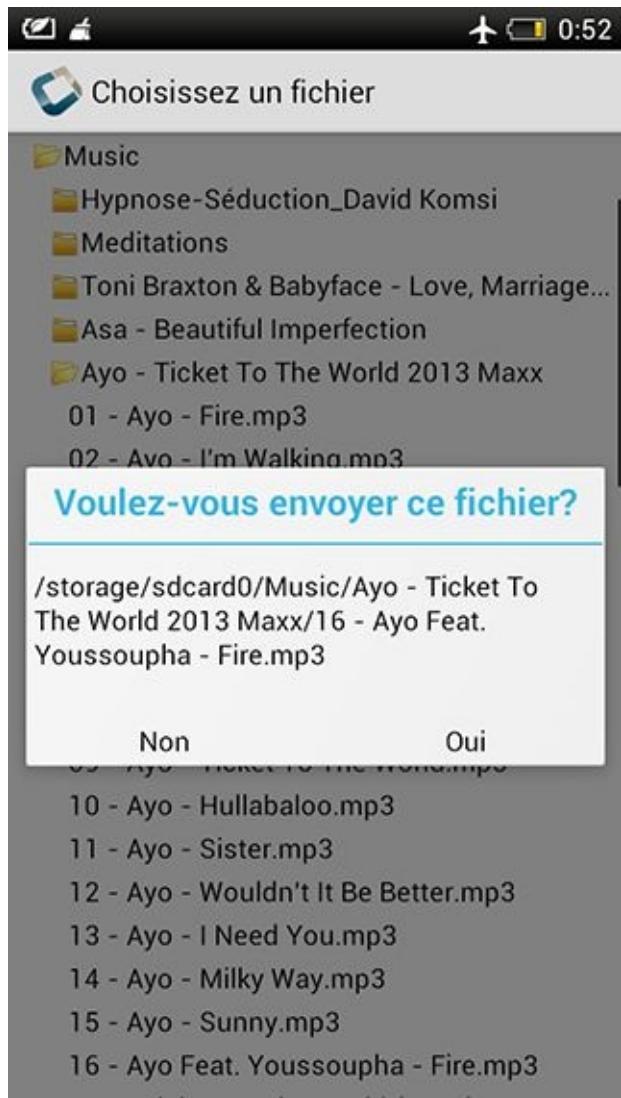
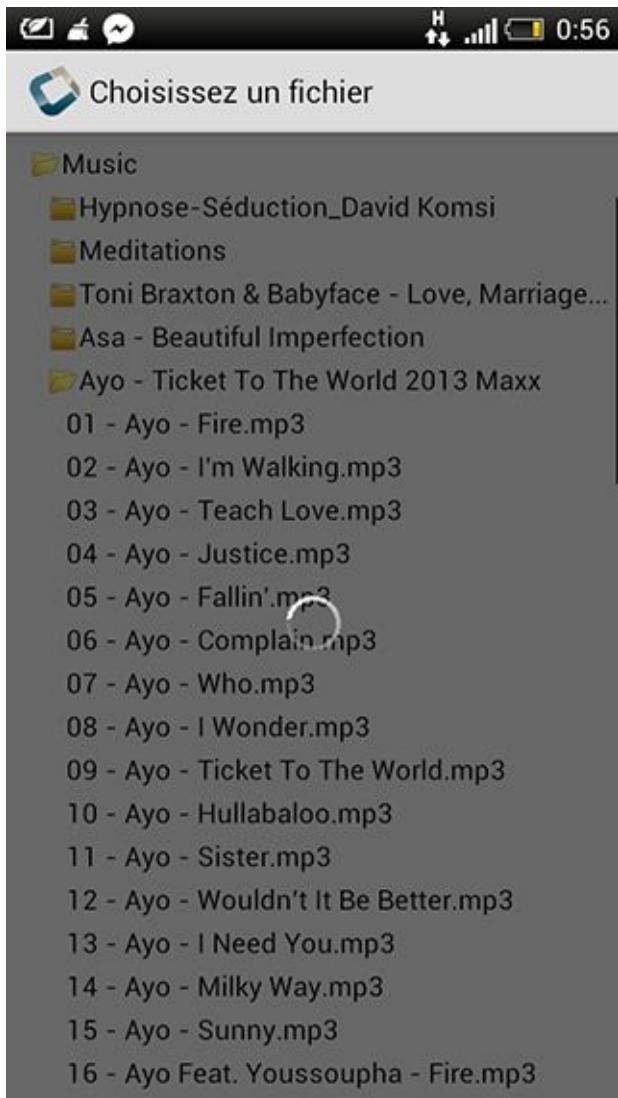


Figure 5.6 : Envoi en cours du fichier choisi



Attention > Pour qu'un serveur accepte des données envoyées par `MultipartRequest`, il faut que ce serveur soit en mesure de traiter les requêtes de type `multipart` qui est un type MIME un peu particulier.

Voyons maintenant un exemple de code PHP simpliste de récupération du fichier uploadé vers le serveur (fichier `upload.php` utilisé dans le code Java).

```
<?php  
$nom_fichier = "audio.mp3";
```

```
❶ move_uploaded_file($_FILES["contenu"]["tmp_name"], "sons/".$nom_fichier); ❷ ?  
>
```

❶ Création d'une variable contenant le nom du fichier (`audio.mp3`) qui sera créé sur le serveur et qui représentera le fichier MP3 envoyé par l'application mobile.

Récupération du fichier MP3 envoyé avec `$_FILES` et déplacement du fichier dans le ❷ dossier `sons` présent sur le serveur. Ce déplacement est fait avec la fonction `move_uploaded_file()`.

1.3. Autres classes dérivées de ConnectionRequest

Dans cette section, nous allons voir deux autres classes dérivées de ConnectionRequest, qui vous aideront aussi à simplifier certaines tâches. Nous ne détaillerons pas leur utilisation car il suffit de comprendre comment fonctionne ConnectionRequest pour bien les utiliser. Il s'agit des classes [RSSService](#) et [GZConnectionRequest](#).

RSSService (et RSSReader)

Comme son nom l'indique, RSSService permet de lire des contenus de flux RSS. Il prend l'URL d'un flux RSS et retourne les données du flux dans un Vector de Hashtable. À utiliser dans la même logique que ConnectionRequest.

Pour ceux qui préfèrent les outils déjà prêts à l'emploi, un composant nommé RSSReader utilise RSSService en interne pour dresser dans un composant de type List le contenu RSS récupéré. L'utilisation de ce composant est simple parce qu'il suffit de l'instancier, d'ajouter l'URL du flux et de l'ajouter à l'interface de votre application. Un clic sur le titre d'un article de la liste l'affichera au complet sur une autre fenêtre. Voici un exemple d'utilisation de RSSReader.

```
Form f=new Form("Actualités");
RSSReader reader=new RSSReader();
❶ reader.setURL("http://www.codenameonefr.com/blog/feed/");
❷ f.addComponent(reader);
```

❶ Création de l'objet RSSReader.

❷ Ajout de l'URL du flux RSS avec la méthode `setURL()`.

Figure 5.7 : Liste des articles renvoyées par le flux

RSS

Interview de Steve Hannah, créateur de CN1M...

Bonjour, J'espère que vous allez bien. ça fait quelques mois que vous n'avez pas eu de mes

Comment utiliser le HTML pour créer les interf...

Salut, Aujourd'hui, je vais vous parler d'un plug-in qui a fait son apparition la semaine

C'est le retour de CodenameOneFr.com. Crée...

Salut, L'attente a été longue, voire très longue, pour ceux d'entre vous qui ont vu disparaître

Mise à jour du document d'initiation (version 1....

La version 1.3 du document d'initiation est disponible avec une page entière consacrée au

Interview du co-concepteur et C.E.O de Coden...

Shai Almog, ce nom vous dit peut-être quelque chose ou peut-être pas, mais derrière ce nom

Nouvelle mise à jour du document d'initiation à ...

Bonjour, Après 2 semaines d'absence, me voici avec 2 nouvelles La version 1.2 du

Nouvelle version du document d'initiation à Cod...

Figure 5.8 : Affichage d'un article

Interview de Steve Hannah, créateur de CN1ML et de

Bonjour, J'espère que vous allez bien. ça fait quelques mois que vous n'avez pas eu de mes nouvelles et c'est parce que j'étais en train de travailler sur le premier livre de Codename One qui arrivera bientôt. Je vous en dirai plus dans un prochain article. Dans l'article précédent, je vous ai parlé de CN1ML [...]

Note > Le rendu de RSSReader est basique donc si vous préférez avoir un visuel personnalisé, utilisez RSSService et ajoutez les données lues aux composants de votre choix sur l'interface.

Attention > Des problèmes de lecture du contenu du flux peuvent subvenir avec des flux ne respectant pas tous les standards du format RSS.

GZConnectionRequest

La classe GZConnectionRequest permet de détecter et de décompresser automatiquement une réponse de requête HTTP si elle est compressée au format GZip. Celui-ci est utilisé par certains serveurs pour la compression de données. Une requête créée avec GZConnectionRequest décomprime les données reçues mais ne cherche pas à récupérer automatiquement des données gzippées. Pour cela, vous devez le préciser dans la requête à travers un header HTTP. Le code ressemblerait à ceci :

```
GZConnectionRequest requete=new GZConnectionRequest() ;  
Requete.addRequestHeader("Accept-Encoding", "gzip");
```

Parmi les méthodes que vous pouvez redéfinir et implémenter, readUnzippedResponse() est très semblable à [readResponse\(\)](#) que nous avons à la [Section 1.1, Téléchargement de données](#).

Pour ce qui est de la compression/décompression des données dans un flux, utilisez les classes GZipInputStream et GZipOutputStream.

1.4. Quelques méthodes de ConnectionRequest

- void addRequestHeader(String key, String value) – Ajoute un header HTTP et sa valeur à la requête. En premier paramètre le nom du header HTTP et en deuxième la valeur à attribuer au header.
- void setTimeout(int time) – Indique le temps d'exécution de la requête. Le paramètre time est en millisecondes.
- void setSilentRetryCount(int silentRetryCount) – En cas d'échec de la connexion, cette méthode indique le nombre de fois où une tentative de reconnexion doit être effectuée automatiquement avant l'abandon. Entrez le nombre de fois en paramètre.
- void setShowInit(Dialog d) – Affiche la boîte de dialogue passée en paramètre une fois que la requête entre dans la file d'attente d'exécution créée par NetworkManager.
- void setDuplicateSupported(boolean duplicate) – Indique si la requête de connexion doit accepter ou non des entrées dupliquées ou similaires.
- boolean pause() – Sert à mettre en pause l'opération en cours. Elle retournera la

valeur false si l'opération est déjà terminée ou est redondante.

- `boolean resume()` – Sert à reprendre l'exécution d'une opération auparavant mise en pause.
- `void addArgument(String key, String value)` – Permet d'ajouter un argument et sa valeur à la requête. En premier paramètre le nom de l'argument, en deuxième la valeur de l'argument.
- `void addArgumentNoEncoding(String key, String value)` – Cette méthode joue le même rôle que `addArgument()` sans encoder les arguments passés en paramètre à la requête. À utiliser pour les valeurs d'arguments déjà encodées.
- `int getResponseCode()` – Retourne le code de réponse HTTP de la requête.

1.5. Quelques méthodes de NetworkManager

- `void addProgressListener(ActionListener listener)` – Ajoute un ActionListener pour être notifié de la progression de l'opération en cours.
- `int getThreadCount()` – Retourne le nombre de threads en cours d'exécution.
- `void killAndWait(ConnectionRequest request)` – Stoppe la requête passée en paramètre.
- `void shutdown()` – Arrête le thread du réseau de manière asynchrone.
- `void shutdownSync()` – Arrête le thread du réseau de manière synchrone (bloquante).

2. Communication avec un service web

Dans cette section, nous allons parler des services web. Avant toute chose, définissons un service web. Un service web est un programme qui agit en quelque sorte comme un pont pour permettre à diverses sources écrites avec des langages de programmation différents (pages web, application desktop, application mobile, etc.) d'échanger entre elles des données centralisées sur un serveur. De manière générale, l'interrogation d'un service web retourne des données au format JSON ou XML.

Note > Si vous n'avez vraiment aucune idée de ce qu'est un service web malgré la définition donnée ci-dessus, faites alors un tour sur le web pour plus d'informations.

2.1. Utilisation d'un service web quelconque

Pour le test de communication avec un service web, j'ai choisi celui du site de stockage d'images en ligne [Flickr](#). L'application que nous allons écrire va permettre d'avoir accès à une liste d'images publiques de Flickr, de les télécharger et de créer un carrousel avec ces images. Pour obtenir ces images, nous aurons seulement besoin de l'URL à interroger (à savoir : http://api.flickr.com/services/feeds/photos_public.gne), d'un mot clé pour spécifier le genre d'images souhaité et du format dans lequel nous voulons recevoir les données envoyées par le serveur.

Note > Si vous souhaitez tester d'autres fonctionnalités de Flickr, sachez que certaines exigent d'être inscrit et de demander une clé d'authentification que vous utiliserez comme argument lors de la création de la requête.

Les images publiques que nous allons chercher seront des images qui correspondront au mot clé saisi par l'utilisateur de l'interface de l'application. Pour le format des données à récupérer, nous optons pour le JSON.

Exemple 5.5 : Utilisation du service web de Flickr pour le téléchargement d'images

```
public class FlickrSlider implements ActionListener {  
  
    private Form current;  
    private Button telechargerImages;  
    private Button afficherImages;  
    private TextField motcle;  
    private Vector imgs;  
    private String racine;  
    private Command retour;  
    private Command quitter;  
    private Form f;  
  
    public void init(Object context) {  
        try {  
            Resources theme = Resources.openLayered("/theme");  
            UIManager.getInstance().setThemeProps(  
                theme.getTheme(theme.getThemeResourceNames()[0]));  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

        } catch(IOException e){
            e.printStackTrace();
    }

}

public void start() {
    if(current != null){
        current.show();
        return;
    }
}

```

❶ f=new Form("Flickr Slider"); f.setLayout(new BoxLayout(BoxLayout.Y_AXIS)); motcle=new TextField(); motcle.setHint("Entrez un mot clé pour la recherche"); telechargerImages=new Button("Télécharger et afficher"); afficherImages=new Button("Afficher les images"); telechargerImages.addActionListener(this); afficherImages.addActionListener(this); quitter=new Command("Quitter"); f.addCommand(quitter); f.addComponent(motcle); f.addComponent(telechargerImages); f.addComponent(afficherImages); f.addCommandListener(this); f.show(); } **public void** stop() { current=Display.getInstance().getCurrent(); } **public void** destroy() { } **void** creerAfficherSlide() **❷** { Form fSlide=new Form("Flickr Slider"); fSlide.setLayout(new BorderLayout()); ImageViewer viewer=new ImageViewer(); DefaultListModel<Image> listeImage=new DefaultListModel<Image>(); try { listeImage.addItem(Image.createImage(racine+"photo0.jpg")); listeImage.addItem(Image.createImage(racine+"photo1.jpg")); listeImage.addItem(Image.createImage(racine+"photo2.jpg")); listeImage.addItem(Image.createImage(racine+"photo3.jpg")); } catch(IOException ex){ Dialog.show("Erreur", "Images non trouvées", "Ok", null); return; } viewer.setImage(listeImage.getItemAt(0)); viewer.setImageList(listeImage); retour=new Command("Accueil"); fSlide.addCommand(retour); fSlide.setBackCommand(retour); fSlide.addComponent(BorderLayout.CENTER, viewer); fSlide.addCommandListener(this); fSlide.show(); } **public void** actionPerformed(ActionEvent evt) { Object obj=evt.getSource(); Command cmd=evt.getCommand(); racine=FileSystemStorage.getInstance().getAppHomePath(); **❸** if(!racine.endsWith("/")){ racine+="/"; } **if**(obj==telechargerImages){ imgs=new Vector(); ConnectionRequest requete=new ConnectionRequest(); **@Override protected void** postResponse() { **❹ if**(Dialog.show("", "Connexion établie. Validez pour télécharger les images", "OK", null)){ **int** nbreUrlImages=imgs.size(); **for**(**int** i=0;i<nbreUrlImages;i++){ Util.downloadUrlToFile((String)imgs.elementAt(i), racine+"photo"+i+".jpg", true); } imgs=null; creerAfficherSlide(); } } **@Override protected void** readResponse(InputStream input) **throws** IOException { **❺** JSONParser parser=new JSONParser(); Map m=parser.parseJSON(**new** InputStreamReader(input)); List items=(List)m.get("items"); **❻ for**(**int** i=0;i<items.size();i++){ Map item=(Map)items.get(i); String urlImage=(String)((Map)item.get("media")).get("m"); imgs.addElement(urlImage); } } }; **❷** requete.setUrl("http://api.flickr.com/services/feeds/photos_public.gne"); requete.setPost(false); **❸** String mCle=motcle.getText(); **if**(mCle.length()==0){

```

Dialog.show("", "Entrez un mot clé", "Ok", null); return; } ⑨
requete.addArgument("tags", mCle); requete.addArgument("format", "json");
NetworkManager.getInstance().addToQueue(requete); InfiniteProgress progress=new
InfiniteProgress(); Dialog d=progress.showInifiniteBlocking();
requete.setDisposeOnCompletion(d); } else if(obj==afficherImages){ ⑩
creerAfficherSlide(); } if(cmd==retour){ f.showBack(); ⑪ } else if(cmd==quitter){
Display.getInstance().exitApplication(); } } }

```

Création de la fenêtre d'accueil, d'une zone de texte pour la saisie d'un mot clé, de deux boutons (l'un pour télécharger et afficher les images, l'autre pour afficher ① uniquement les fichiers déjà téléchargés), du menu Quitter pour fermer l'application. Ajout des composants créés à la fenêtre, abonnement à l'ActionListener (représenté ici par notre classe) pour la réception des événements, affichage de la fenêtre.

② Création de la méthode `creerAfficherSlide()`. Elle crée une seconde fenêtre, un ImageViewer pour charger les images déjà téléchargées. Le tout sera ensuite affiché.

③ Récupération du dossier racine de l'application pour y sauvegarder les images à télécharger.

Si le premier bouton avec le texte Télécharger et afficher est activé, nous créons la requête de connexion. Dans la méthode `postResponse()` de la requête que nous implémentons ici, nous affichons d'abord une boîte de dialogue pour confirmer la connexion réussie au serveur. Il s'agit en réalité de la fin de la récupération des ④ données JSON envoyées par le serveur et récupérées dans la méthode `readResponse()`. Une fois la boîte de dialogue fermée, nous téléchargeons avec `downloadUrlToFile()` les images dont les URL ont été ajoutées dans un tableau (Vector) dans la méthode `readResponse()`.

Ici, nous sommes dans la méthode `readResponse()` et nous récupérons les données envoyées par le serveur. Ces données étant au format JSON, nous utilisons la classe ⑤ `JSONParser` pour parser ces données et les sauvegarder dans un objet de type Map. La ligne avec le code `List items=(List)m.get("items")` nous permet de récupérer tous les éléments à l'intérieur du bloc `items` dans un objet List.

⑥ Dans cette boucle, nous récupérons l'URL de l'image de chaque élément à l'intérieur du bloc `items` et nous les ajoutons à un tableau Vector nommé `imgs`.

Ajout de l'URL à interroger et choix de la méthode GET pour indiquer la récupération ⑦ de données.

Étant donné que nous avons besoin d'un mot clé pour effectuer la recherche d'images, nous vérifions si l'utilisateur a entré quelque chose dans ce champ. Si ce n'est pas le ⑧

cas alors nous affichons une boîte de dialogue en guise de rappel.

Ajout des arguments tags et format à la requête. Ici, il s'agit du mot clé et du format dans lequel nous voulons recevoir les données. Ce mot clé sera celui qui est entré par ❾ l'utilisateur et le format sera le JSON. Une fois le paramétrage terminé, nous ajoutons la requête à la file d'exécution et nous créons une fenêtre de progression pour indiquer l'exécution en cours de la requête.

Si le deuxième bouton avec le texte Afficher les images est activé, appeler la méthode ❿ créerAfficherSlide() pour créer et afficher le carrousel avec les images déjà téléchargées.

Si le menu retour avec le texte Accueil est activé alors nous affichons la première ❻ fenêtre. showBack() est semblable à show() et permet d'afficher un Form dans le sens inverse de la transition qui a fait apparaître la fenêtre courante, laquelle est ici celle contenant le carrousel.

Figure 5.9 : Téléchargement d'images sur Flickr et création d'un carrousel (vidéo)



2.2. Communication avec une base de données distante

Maintenant que nous savons comment communiquer avec un service web, nous allons en créer un nous-mêmes pour communiquer avec une base de données en ligne. Supposons que vous voulez créer une application qui n'a pas besoin d'une base de données embarquée, mais plutôt d'une base de données en ligne sur un serveur quelconque pour enregistrer les données. Par défaut, il n'est pas possible d'utiliser les classes et méthodes vues au chapitre [Persistance des données](#). Une application mobile n'a pas cette possibilité. Il faudra donc créer et utiliser un service web qui servira d'intermédiaire entre l'application et la base de données distante.

Note > *Utiliser un service web n'est pas la seule méthode pour échanger entre une application mobile et une base de données distante. Vous pourriez aussi utiliser les sockets, mais le service web reste le moyen le plus simple et le plus approprié.*

Comme pour l'exemple d'utilisation de MultipartRequest, nous utiliserons le PHP pour concevoir un service web élémentaire fondé sur l'architecture REST. Les outils dont vous avez besoin pour exécuter du code PHP ont été mentionnés dans l'introduction de ce chapitre. *Si vous n'utilisez pas ou n'aimez pas le PHP, alors écrivez un code équivalent dans votre langage web préféré.*

Note > *REST (Representational State Transfer) n'est pas un protocole mais un style d'architecture basé sur le HTTP. Il permet d'accéder à une ressource distante à travers les opérations proposées par le protocole HTTP. Il s'agit des opérations de lecture (GET), d'écriture (POST), de modification (PUT) et de suppression (DELETE).*

L'exemple que nous allons concevoir est simple. Il s'agira d'une interface pour effectuer une inscription en ligne. Celle-ci sera juste composée de deux zones de texte (identifiant et mot de passe) et d'un bouton pour valider l'inscription. Un clic sur ce bouton va récupérer les données saisies par l'utilisateur et les envoyer au service web PHP sur le serveur. Ce service web récoltera les données envoyées par l'application, les enregistrera dans la base de données en ligne et renverra un message à l'application. Le message envoyé sera au format JSON et l'application se chargera de le récupérer et de l'afficher dans une boîte de dialogue. Le code du service web est minimal et ne contient aucun contrôle. Pensez à le sécuriser avec plus de contrôles si vous le déployez sur un serveur en ligne.

Création d'un simple service web REST en PHP

Avant de passer au code du service web, créez une base de données en ligne avec trois champs. Un champ `id` auto-incrémental, un champ `identifiant` et un dernier champ `motdepasse`. Dans notre exemple, la base est nommée `resttestbdd` et la table `utilisateurs`.

Passons maintenant au code du service web PHP (fichier `inscription.php`). La base de données MySQL utilisée ici ne se trouve pas en ligne, mais en local sur la machine du développeur. C'est la raison pour laquelle `localhost` sera utilisé comme adresse de connexion dans le code que nous allons écrire.

Note > *Ce livre n'étant pas un livre d'apprentissage du PHP, je ne m'attarderai pas sur le détail de chaque fonction du code. Je me contenterai d'une explication globale.*

Exemple 5.6 : Fichier PHP du service web REST de communication avec la base de données

```
<?php
```

```
❶ $connect=mysql_connect("localhost", "root", ""); mysql_select_db('resttestbdd', $connect); if($_SERVER['REQUEST_METHOD'] == "POST"){ ❷ ❸  
$login=mysql_real_escape_string($_POST['identifiant']);  
$password=mysql_real_escape_string($_POST['motdepasse']);  
$requete=mysql_query("INSERT INTO utilisateurs VALUES (NULL, '$login', '$password')"); ❹ if($requete){ $data=array("reponse" => "Inscription effectuée.  
Bienvenue!"); ❺ } else { $data=array("reponse" => "Echec d'inscription. Réessayez  
SVP"); ❻ } } else{ $data=array("reponse" => "Méthode non acceptée"); } ❼  
header('Content-type: application/json'); echo json_encode($data); ?>
```

Attention > Pour le test, le mot de passe est stocké en clair dans la base de données dans notre code. En production, pensez à le crypter avant de l'ajouter à la table.

- ❶ Connexion à MySQL suivie du choix de la base de données qui nous intéresse.
- ❷ Vérification du type de la méthode d'envoi utilisée par l'application mobile. S'il s'agit de POST alors nous continuerons avec la suite du programme.
- ❸ Récupération des deux données (identifiant et mot de passe) envoyées par l'application.
- ❹ Exécution de la requête d'insertion dans la base de données.
- ❺ Création de la variable data qui est un tableau contenant la confirmation à envoyer à l'application si la requête est bien exécutée.
- ❻ Création de la variable data qui est un tableau contenant le message d'erreur à envoyer à l'application si l'exécution de la requête échoue.
- ❼ Encodage du contenu de la variable data au format JSON suivi de son envoi vers l'application.

Consommation du service web créé

Passons maintenant au code de l'application proprement dite.

Exemple 5.7 : Application d'envoi des données saisies au service web pour la base de données

```

public class CN1BDDDistant implements ActionListener {

    private Form current;
    private Command quitter;
    private TextField login;
    private TextField password;
    private Button inscription;

    public void init(Object context) {
        try {
            Resources theme = Resources.openLayered("/theme");
            UIManager.getInstance().setThemeProps(
                theme.getTheme(theme.getThemeResourceNames()[0]));
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    public void start() {
        if(current != null){
            current.show();
            return;
        }
        Form f = new Form("Base de données distante");
        BorderLayout mainLayout=new BorderLayout();

        mainLayout.setCenterBehavior(BorderLayout.CENTER_BEHAVIOR_CENTER_ABSOLUTE);
        f.setLayout(mainLayout);

        Container mainContainer=new Container(new
        BoxLayout(BoxLayout.Y_AXIS));

```

❶ login=new TextField(); login.setHint("Entrez votre identifiant"); password=new TextField(); password.setHint("Entrez votre mot de passe"); password.setConstraint(TextField.PASSWORD); inscription=new Button("Je m'inscris"); mainContainer.addComponent(login); mainContainer.addComponent(password); mainContainer.addComponent(inscription); quitter=new Command("Quitter"); ❷ ❸ f.addComponent(BorderLayout.CENTER, mainContainer); f.addCommand(quitter); inscription.addActionListener(this); f.addCommandListener(this); f.show(); } **public void** stop() { current = Display.getInstance().getCurrent(); } **public void** destroy() { } **public void** actionPerformed(ActionEvent evt) { **if**(evt.getSource()==inscription){ ConnectionRequest requete=new ConnectionRequest(){ ❹ Map data; ❺ @Override **protected void** postResponse() { Dialog.show("",(String)data.get("reponse"), "Ok", null); ❻ } @Override **protected void** readResponse(InputStream input) **throws** IOException { ❻ JSONParser parser=new JSONParser(); data=parser.parseJSON(new InputStreamReader(input)); } }; ❽ requete.setUrl("http://localhost/inscription.php"); requete.setPost(true); requete.addArgument("identifiant", login.getText()); requete.addArgument("motdepasse", password.getText()); NetworkManager.getInstance().addToQueue(requete); InfiniteProgress progress=new InfiniteProgress(); Dialog d=progress.showInfiniteBlocking();

```
requete.setDisposeOnCompletion(d); } ❾ if(evt.getCommand()==quitter){  
Display.getInstance().exitApplication(); } } }
```

- ❶ Création des champs de l'identifiant, du mot de passe, du bouton d'inscription. Ajout de ces composants à un Container qui sera placé au milieu de la fenêtre Form.
- ❷ Création d'un menu quitter pour fermer l'application.
- ❸ Ajout du Container contenant les zones de texte de l'identifiant et du mot de passe puis du bouton d'inscription à la fenêtre Form. Ajout du menu quitter à Form puis abonnement du bouton inscription et du Form à l'ActionListener pour la gestion des événements.
- ❹ Création de la requête HTTP.

- ❺ Déclaration d'un objet de type Map pour stocker les données JSON renvoyées par le service web.
- ❻ Affichage de la réponse envoyée par le serveur. Cette réponse est la valeur de la clé reponse que nous récupérons avec la méthode get() de la classe Map.

- ❼ Instanciation de la classe JSONParser dans le but de lire les données JSON envoyées par le serveur. La méthode parseJSON() nous permet de parser les données reçues et les retourne sous forme d'un objet de type Map.

- ❽ Ajout de l'URL, choix de la méthode POST pour l'envoi, ajout des arguments identifiant et motdepasse à la requête. Les valeurs de ces arguments sont les données saisies par l'utilisateur et récupérées avec la méthode getText() de la classe TextField. Ajout de la requête à la file d'exécution et création d'une fenêtre d'attente pendant l'exécution de la requête.

- ❾ Si le menu Quitter est activé alors fermer l'application en faisant appel à la méthode exitApplication() du singleton Display.

Une sortie console du contenu de la variable data (genre System.out.println(data)) nous afficherait un contenu JSON sous cette forme :

```
{reponse=Inscription effectuée. Bienvenue!}
```

Il ne vous reste qu'à aller vérifier dans la base de données en ligne si les données envoyées sont bien présentes.

Figure 5.10 : Saisie des données à envoyer au service web pour la base de données

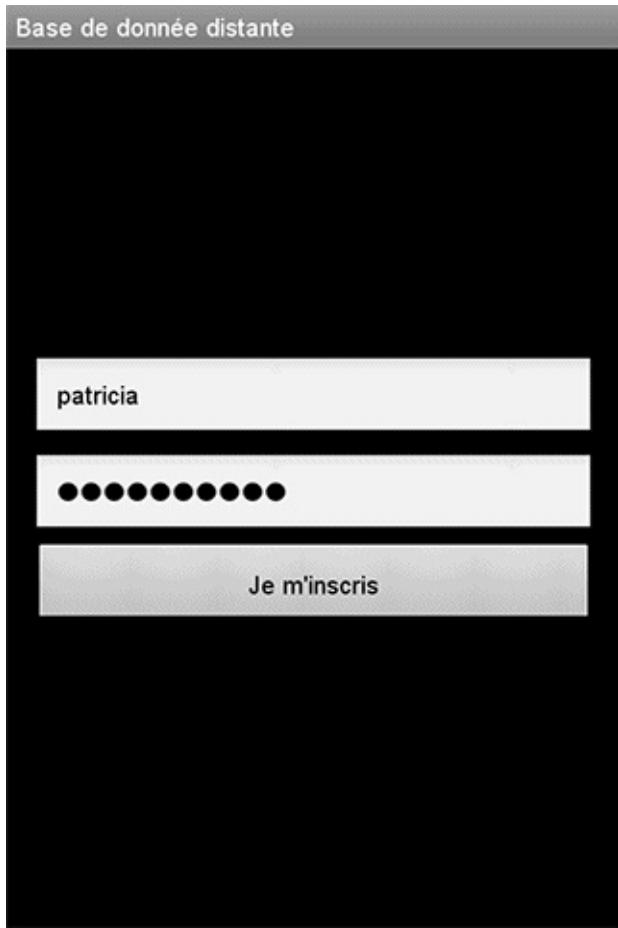


Figure 5.11 : Affichage de la réponse envoyée par le service web



3. Network Monitor (aide au débogage)

Le Network Monitor est une application intégrée au simulateur. Elle permet de suivre l'exécution d'une requête à travers le réseau. Cela peut aider à trouver des bugs glissés dans le code. Cette application est vraiment pratique donc utilisez-la pour optimiser et savoir exactement ce qui se passe durant la communication réseau de votre application avec un serveur. Pour la lancer, sélectionnez l'entrée Network Monitor dans le menu Simulate/Network du simulateur (voir [Figure 5.12](#)). La [Figure 5.13](#) montre l'interface de cette application.

Figure 5.12 : Accès au Network Monitor

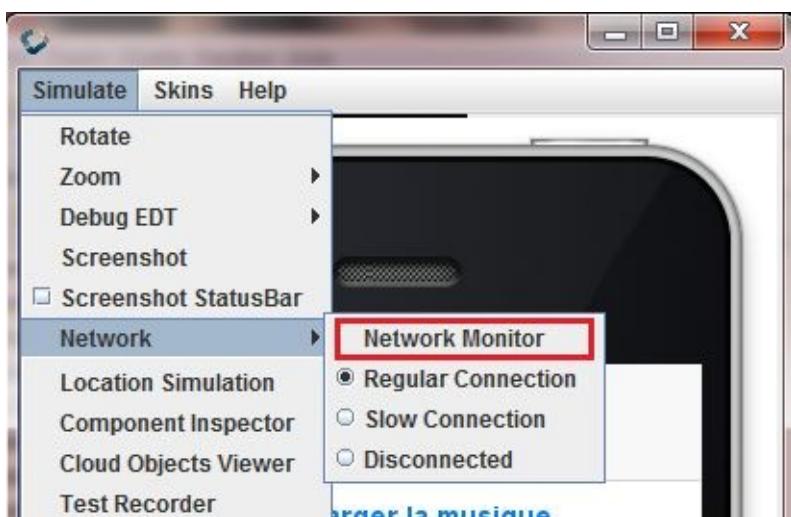
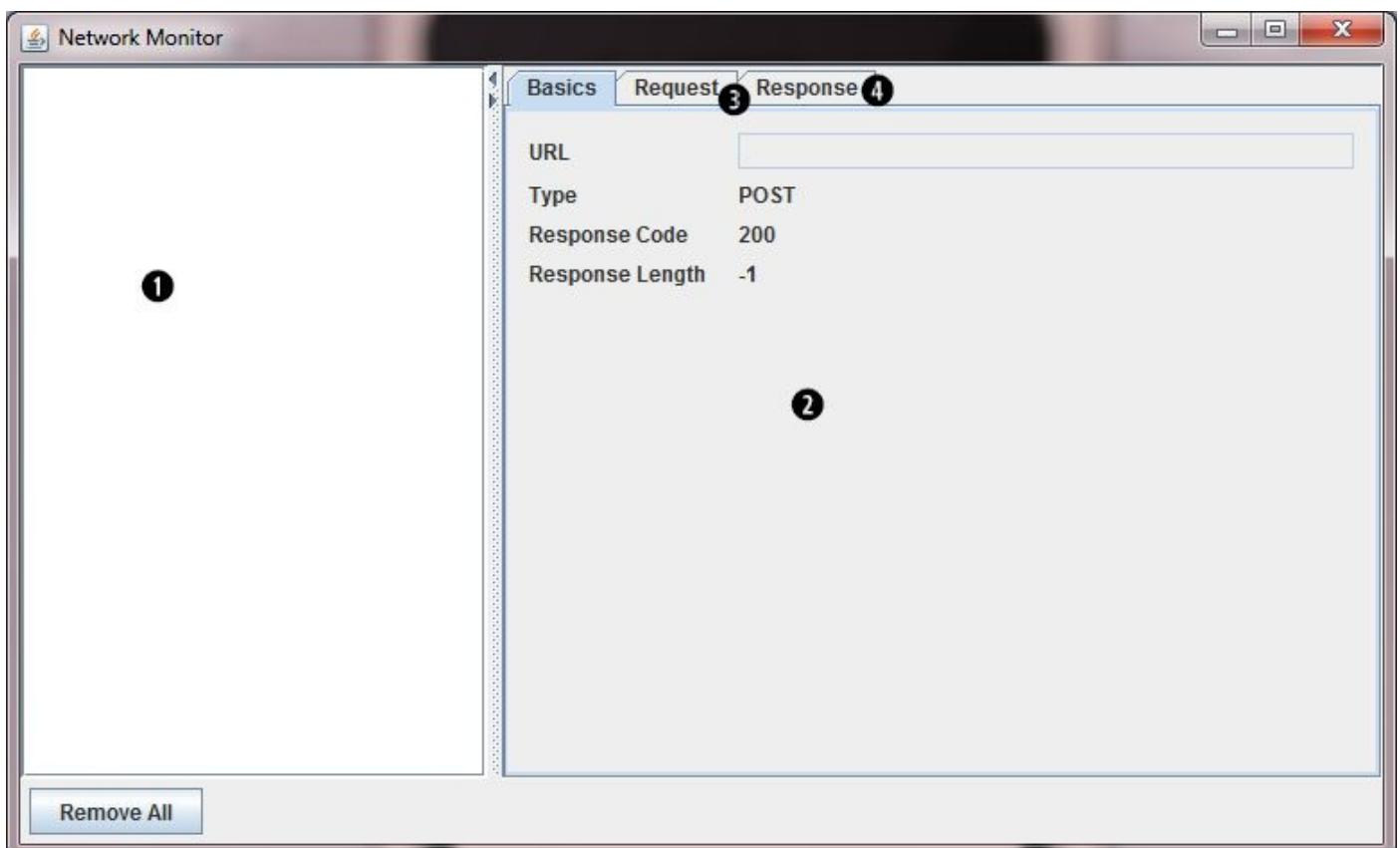


Figure 5.13 : Les différentes zones du Network Monitor



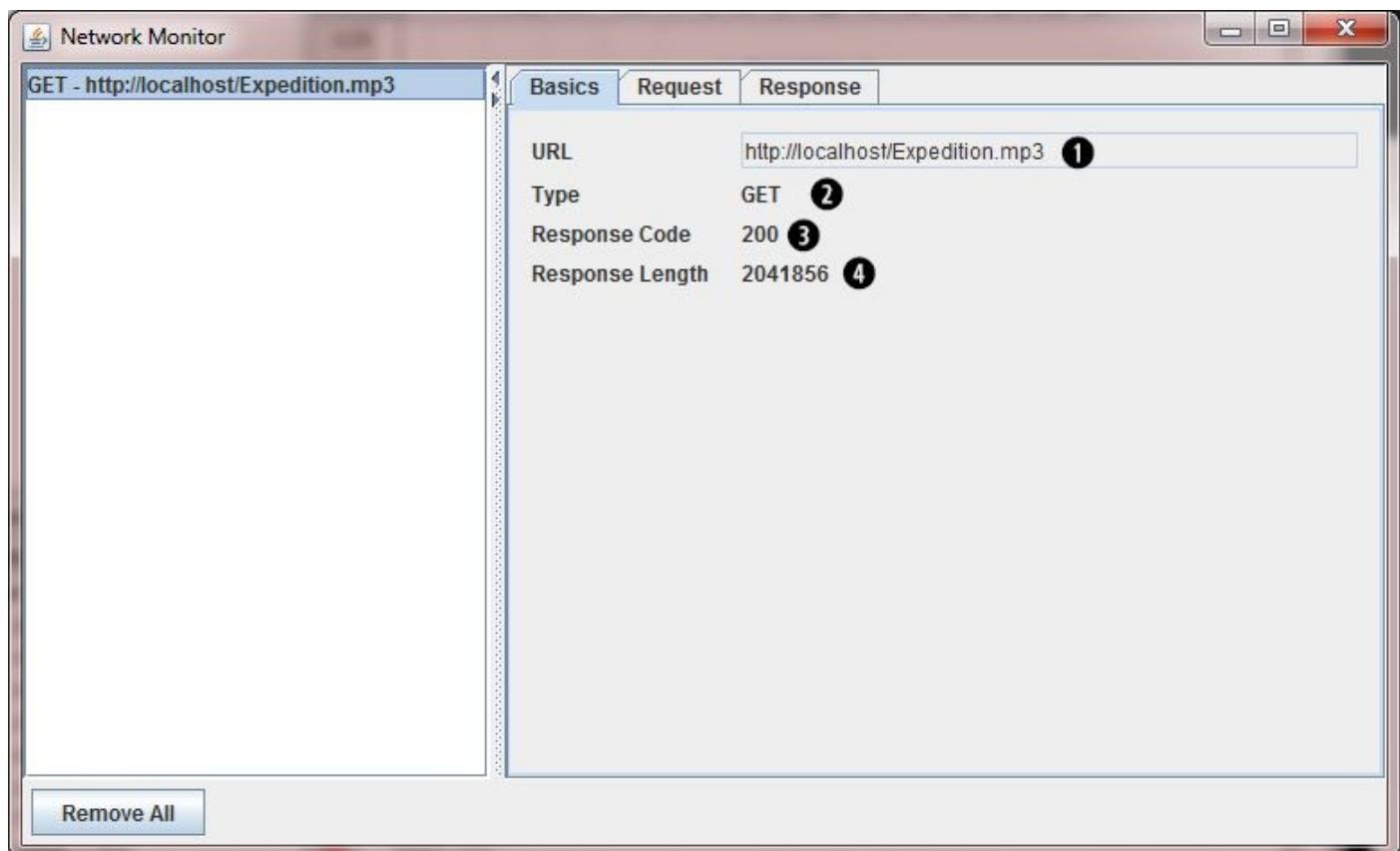
La zone ❶ affiche la liste des requêtes qui sont lancées. Sous l'onglet Basics ❷, vous avez des informations de base sur la requête sélectionnée. L'onglet Request ❸ affiche des informations sur la requête en elle-même et l'onglet Response ❹ sur les données retournées par l'exécution de la requête.

Pour tester cette application, reprenons l'[Exemple 5.1](#), qui nous permettait de télécharger un fichier MP3 depuis un serveur en ligne. Exécutez-le dans le simulateur et lancez le Network Monitor. Cliquez enfin sur le bouton Télécharger la musique.

Les [Figure 5.14](#) et [Figure 5.15](#) montrent les données affichées par le Network Monitor à la fin du téléchargement du fichier. Les valeurs des champs Response Code, Response Length, Response Body et Response Headers seront différentes si vous les examinez alors que le téléchargement est en cours.

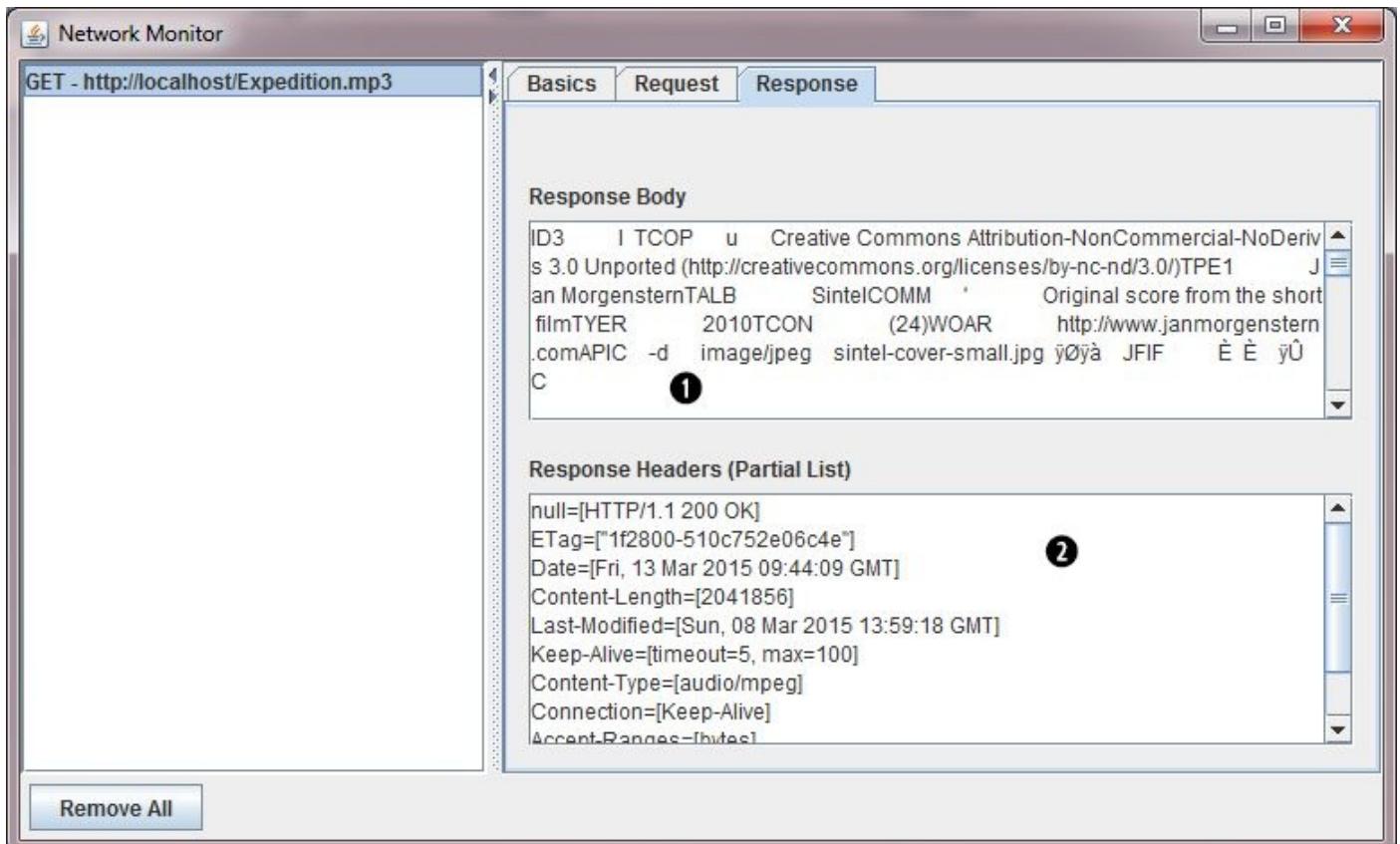
Sous l'onglet Basics de la [Figure 5.14](#), nous voyons en clair l'URL de la requête ❶, son type ❷, le code HTTP retourné ❸ (le code 200 veut dire que tout s'est bien passé) et enfin la taille en octets des données retournées par la requête ❹.

Figure 5.14 : Contenu de l'onglet Basics à la fin de l'exécution de la requête



Sous l'onglet Response de la [Figure 5.15](#), nous voyons les données binaires de la réponse retournée par la requête dans la zone Response Body ❶ et les headers de la réponse de la requête dans la zone Response Headers ❷.

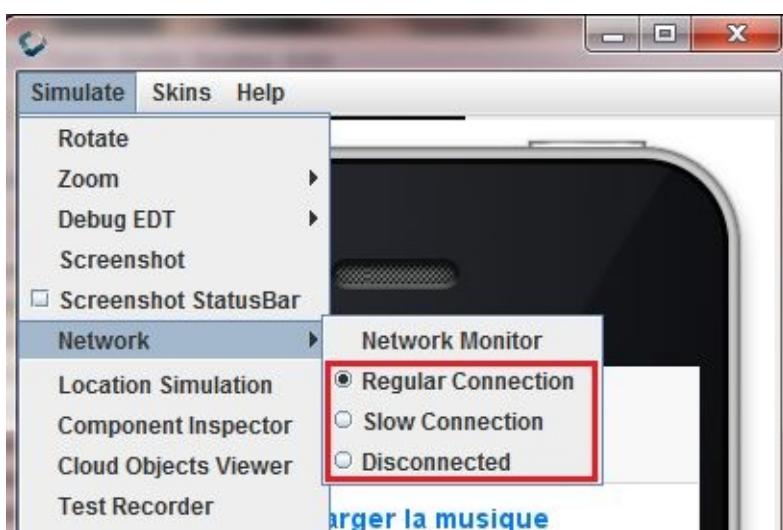
Figure 5.15 : Contenu de l'onglet Response à la fin de l'exécution de la requête



Une fois que vous commencerez à utiliser cet outil, vous y prendrez goût et vous ne pourrez plus vous en passer tant il est pratique.

Pour finir, j'aimerais attirer votre attention sur les autres contenus du sous-menu Network. Il s'agit d'options qui permettent de choisir le niveau de la vitesse de la connexion que vous voulez utiliser pour tester vos applications (voir la [Figure 5.16](#)) : Regular Connection pour une connexion avec le débit que vous avez par défaut ; Slow Connection pour un débit plus lent au niveau du simulateur ; Disconnected permet de couper la connexion toujours au niveau du simulateur.

Figure 5.16 : Réglage du débit de la connexion



Codename One Designer

Ce chapitre est l'un des plus importants de ce livre pour la seule et simple raison que vous ne pourrez vous passer de l'un ou l'autre des éléments présentés ici, à commencer par le fichier de ressources contenu dans chaque projet qui est géré par le Codename One Designer. Nous allons voir comment l'utiliser pour [créer des thèmes](#) pour vos applications, pour concevoir leurs interfaces graphiques avec l'[éditeur graphique](#) plutôt qu'avec du code comme nous l'avons fait jusqu'ici, [gérer leur traduction](#) dans d'autres langues, modifier les [polices de caractères](#) et bien d'autres choses utiles.

1. Présentation du Codename One Designer

Codename One Designer est un logiciel fourni avec le plug-in de Codename One. Ce logiciel permet de manipuler un fichier d'extension .res (le fichier de ressources) qui est un fichier binaire pouvant contenir des thèmes, des images, des dessins d'interfaces, des fichiers d'extensions quelconques, des textes de traduction. Ce fichier est ajouté par défaut à chaque projet et il suffit de l'ouvrir pour lancer le Codename One Designer. Étant donné qu'un fichier .res peut contenir divers types de données qui peuvent s'avérer confidentielles, il est possible de le [protéger par un mot de passe](#) pour éviter qu'il soit ouvert par une tierce personne. Sachez aussi que la taille et la consommation mémoire de vos applications dépendront en grande partie de la taille de ce fichier.

Pour ouvrir le fichier de ressources d'un projet avec le Designer, cliquez droit sur le fichier nommé theme.res dans l'explorateur de fichiers de votre environnement de développement puis sélectionnez Open Res File (voir [Figure 6.1](#)). La [Figure 6.2](#) montre l'interface du Designer.

Note > *Le fichier de ressources est le fichier theme.res de votre projet. C'est le nom qu'il porte par défaut dans chaque projet que vous allez créer mais vous pouvez modifier ce nom si vous le voulez. Vous pouvez aussi créer d'autres fichiers de ressources parce qu'il n'y a pas de limite au nombre que vous pouvez utiliser dans un projet.*

Figure 6.1 : Ouverture d'un fichier de ressources

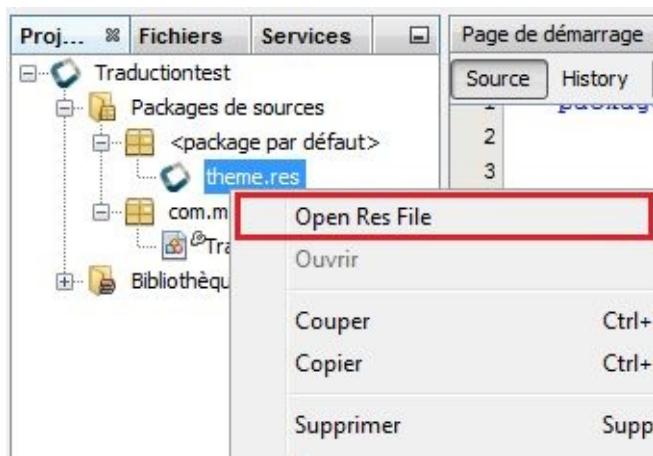
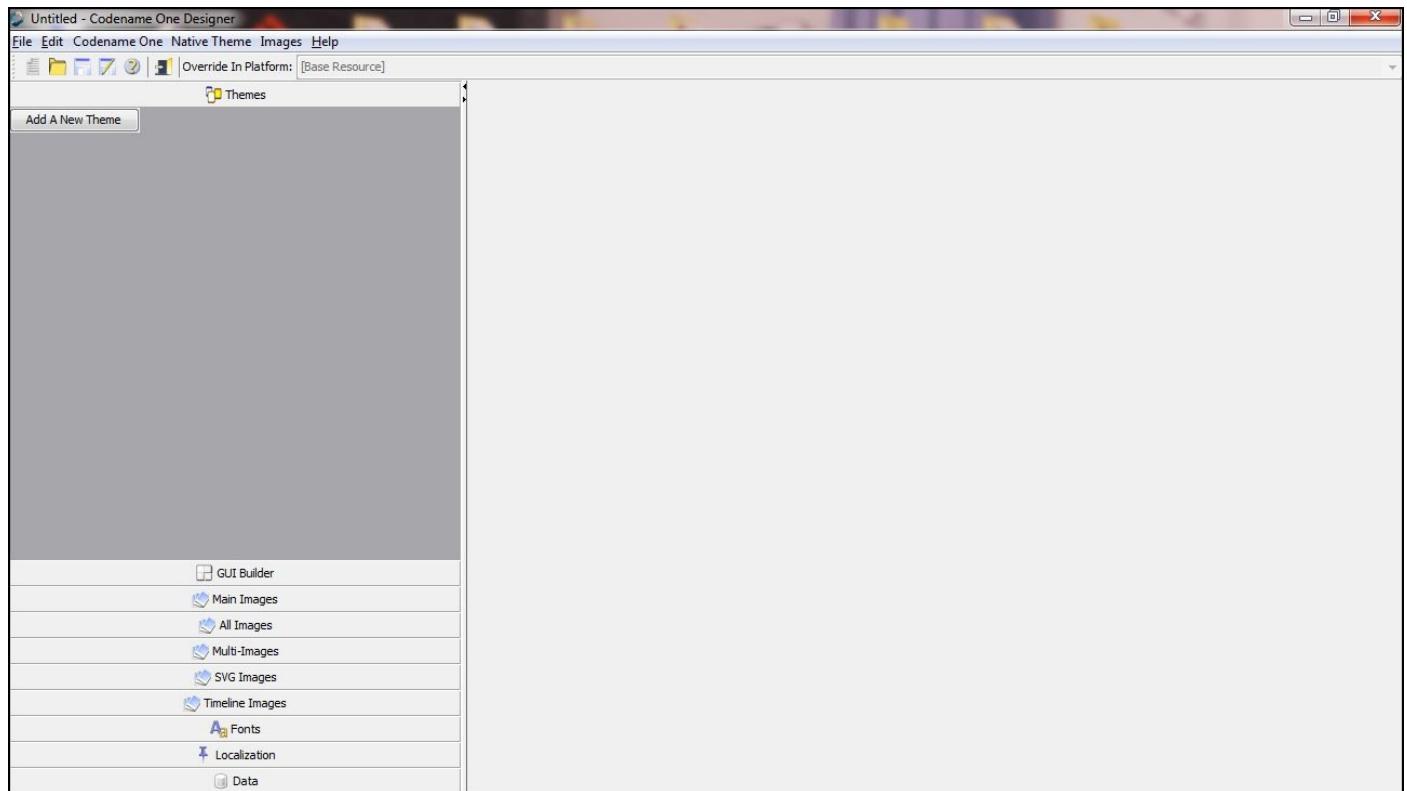


Figure 6.2 : Interface du Codename One Designer



2. Images

Le designer de Codename One permet d'ajouter et de gérer différents types d'images depuis le fichier de ressources. Jusqu'ici, nous avons utilisé la méthode statique `createImage()` de la classe `Image` pour charger les images déposées dans le dossier `SRC` du projet. Dans cette section, nous verrons comment les ajouter au fichier de ressources et comment les charger.

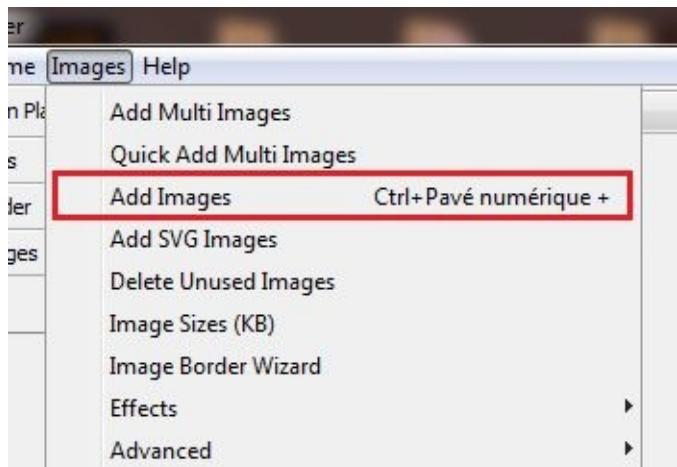
2.1. Images standards

Codename One accepte les images aux formats JPG, PNG ou GIF. Le format GIF n'étant pas supporté sur certains appareils mobiles, il vaut mieux privilégier le JPG et le PNG.

Ajouter une image

Pour ajouter une image au fichier de ressources, allez dans le menu `Images` et cliquez sur `Add Images` (voir [Figure 6.3](#)). Choisissez ensuite une image sur l'ordinateur et validez.

Figure 6.3 : Ajout d'images depuis le menu Images



L'autre manière d'aboutir au même résultat est d'aller sous l'onglet `Main Images`, de cliquer sur le bouton `Add Images`, de sélectionner l'image et de valider (voir [Figure 6.4](#)).

Figure 6.4 : Ajout d'images depuis l'onglet Main Images

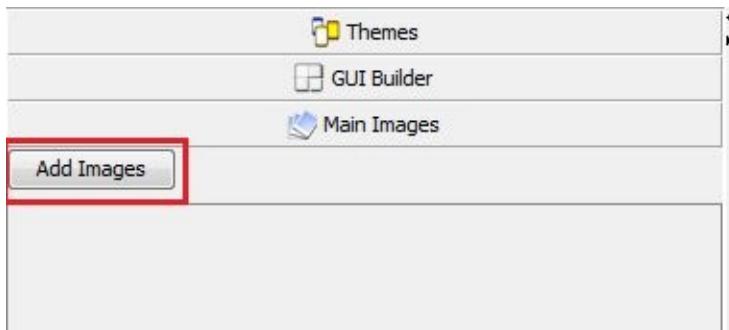
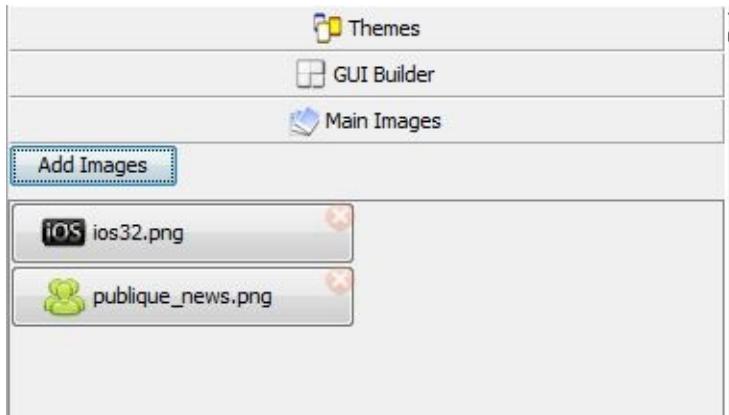


Figure 6.5 : Affichage des images ajoutées



Récupérer une image depuis un fichier de ressources

Une fois les images ajoutées et les modifications enregistrées, il faut utiliser un objet de type Resources pour y accéder. Dans notre cas, nous utiliserons l'objet theme déjà utilisé dans la méthode `init()` pour accéder à une image. La méthode que nous utiliserons est `getImage()` qui prend en paramètre le nom du fichier image à récupérer et son extension.

Voici un code qui récupère une image et l'associe à un Label pour l'affichage.

```
private Resources theme;  
//...  
Image img=theme.getImage("ios32.png");  
Label l=new Label(img);  
//...
```

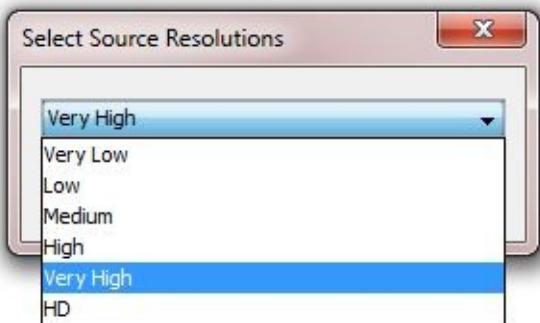
2.2. Multi-images

Un problème courant en développement d'applications mobiles est l'adaptation des images aux diverses résolutions d'écran. La manière classique de le résoudre est de créer manuellement plusieurs versions d'une même image à des tailles différentes et de préciser celle qui sera utilisée en fonction de la résolution de l'écran de l'appareil. Heureusement pour nous, Codename One Designer fournit un outil pour générer automatiquement ces

images. C'est ce qu'il appelle des *multi-images*. Ainsi, au lieu d'avoir une image de même taille sur un petit et un grand écran, l'API de Codename One choisira automatiquement la taille appropriée de cette image.

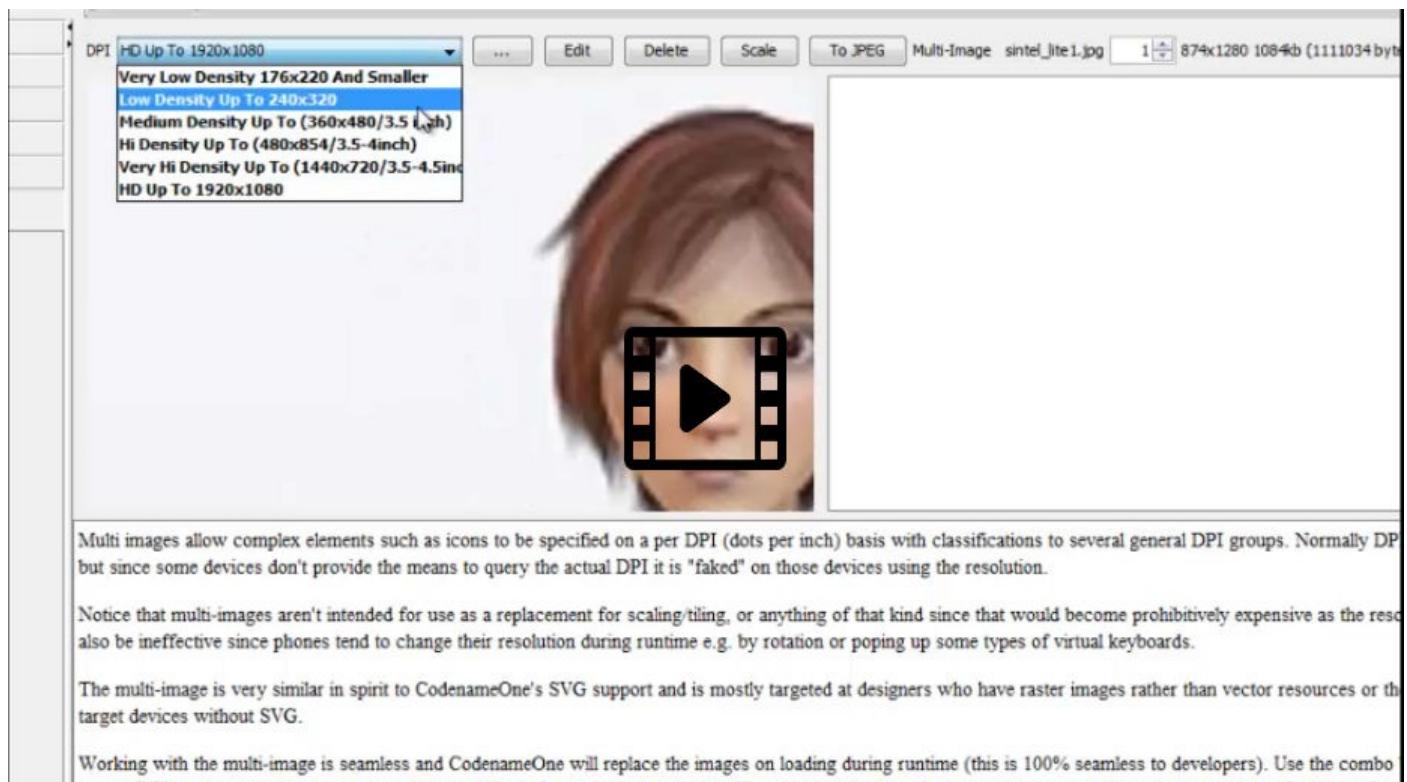
Pour créer une multi-image, allez dans le menu Images puis cliquez sur Quick Add Multi Images. Choisissez une image sur l'ordinateur et validez. Une nouvelle fenêtre s'affiche (voir la [Figure 6.6](#)).

Figure 6.6 : Choix de la résolution de base de l'image



Dans la liste déroulante, sélectionnez la résolution qui vous semble correspondre à la taille de l'image choisie. Une fois ce choix effectué, les versions correspondant aux résolutions inférieures et supérieures à celle que vous avez sélectionnée seront générées automatiquement pour vous. Il y a au total six résolutions : très basse (Very Low), basse (Low), moyenne (Medium), élevée (High), très élevée (Very High), haute définition (HD). La vidéo de la [Figure 6.7](#) montre un exemple de création d'une multi-image.

Figure 6.7 : Crédit d'une multi-image (vidéo)



Multi images allow complex elements such as icons to be specified on a per DPI (dots per inch) basis with classifications to several general DPI groups. Normally DP but since some devices don't provide the means to query the actual DPI it is "faked" on those devices using the resolution.

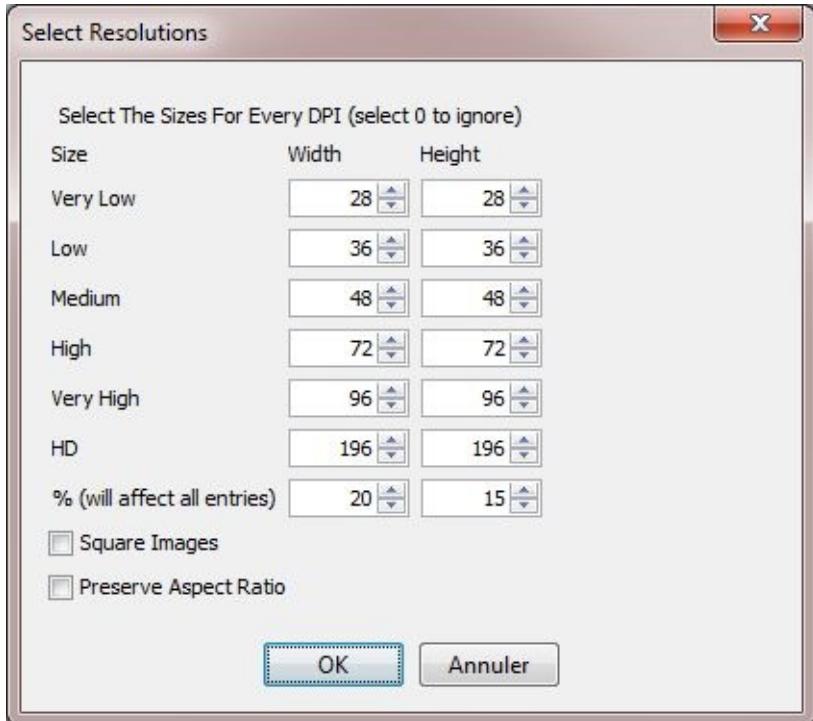
Notice that multi-images aren't intended for use as a replacement for scaling/tiling, or anything of that kind since that would become prohibitively expensive as the reso also be ineffective since phones tend to change their resolution during runtime e.g. by rotation or popping up some types of virtual keyboards.

The multi-image is very similar in spirit to CodenameOne's SVG support and is mostly targeted at designers who have raster images rather than vector resources or the target devices without SVG.

Working with the multi-image is seamless and CodenameOne will replace the images on loading during runtime (this is 100% seamless to developers). Use the combo

L'entrée Quick Add Multi Images du menu Images définit la taille des résolutions à notre place. Si vous souhaitez définir ces tailles par vous-même alors choisissez plutôt l'entrée Add Multi Images qui affichera la fenêtre de la [Figure 6.8](#) et sur laquelle vous pouvez entrer manuellement les valeurs que sous souhaitez pour chacune des résolutions disponibles.

Figure 6.8 : Choix manuel des valeurs de chacune des résolutions



Une fois que les modifications sont enregistrées, faites appel à `getImage()` de Resources pour accéder à l'image.

2.3. Images SVG

Comme nous l'avons vu à la section précédente, les multi-images permettent de supporter diverses résolutions d'écran. Je conseille leur utilisation pour répondre à cette problématique. Toutefois vous pouvez aussi utiliser des images vectorielles au format SVG. L'accès à ces images se fait pareillement avec la méthode `getImage()` de Resources. Si le fichier SVG est déposé dans le dossier SRC du projet au lieu d'être dans le fichier de ressources, utilisez plutôt la méthode `Image.createSVG()` pour le charger.

Toutes les plateformes ne supportent pas le format SVG. Pour ces plateformes, Codename One génère des fichiers PNG multi-images (section précédente) qui seront utilisés à la place des fichiers SVG.

L'ajout d'un fichier SVG dans le Designer se fait à l'aide du menu Images/add SVG Images. Vous pouvez faire la même chose en cliquant sur le bouton Add Images de l'onglet SVG Images depuis la fenêtre principale.

2.4. Images non utilisées et taille des images

Voici deux fonctionnalités pratiques du menu Images :

- Delete Unused Images qui permet de supprimer les images non utilisées dans le GUI Builder et au niveau du (ou des) thème(s) ;
- Images Sizes(KB) qui permet d'afficher en kilo-octets la taille de toutes les images ajoutées au fichier de ressources.

3. Fichiers divers

Il est possible d'ajouter des fichiers de n'importe quel format à un projet dans le but d'utiliser leur contenu dans le code de l'application. Pour cela, deux choix se présentent à nous. Soit déposer les fichiers dans le dossier SRC du projet, soit les insérer à l'intérieur du fichier de ressources. C'est ce dernier cas que nous allons voir dans cette section.

Ouvrez le fichier de ressources de votre projet et allez sous l'onglet Data. Cliquez ensuite sur le bouton Add Data File puis choisissez sur votre ordinateur le fichier à ajouter (voir la [Figure 6.9](#)). Enregistrez ensuite les modifications.

Figure 6.9 : Choix du fichier à ajouter

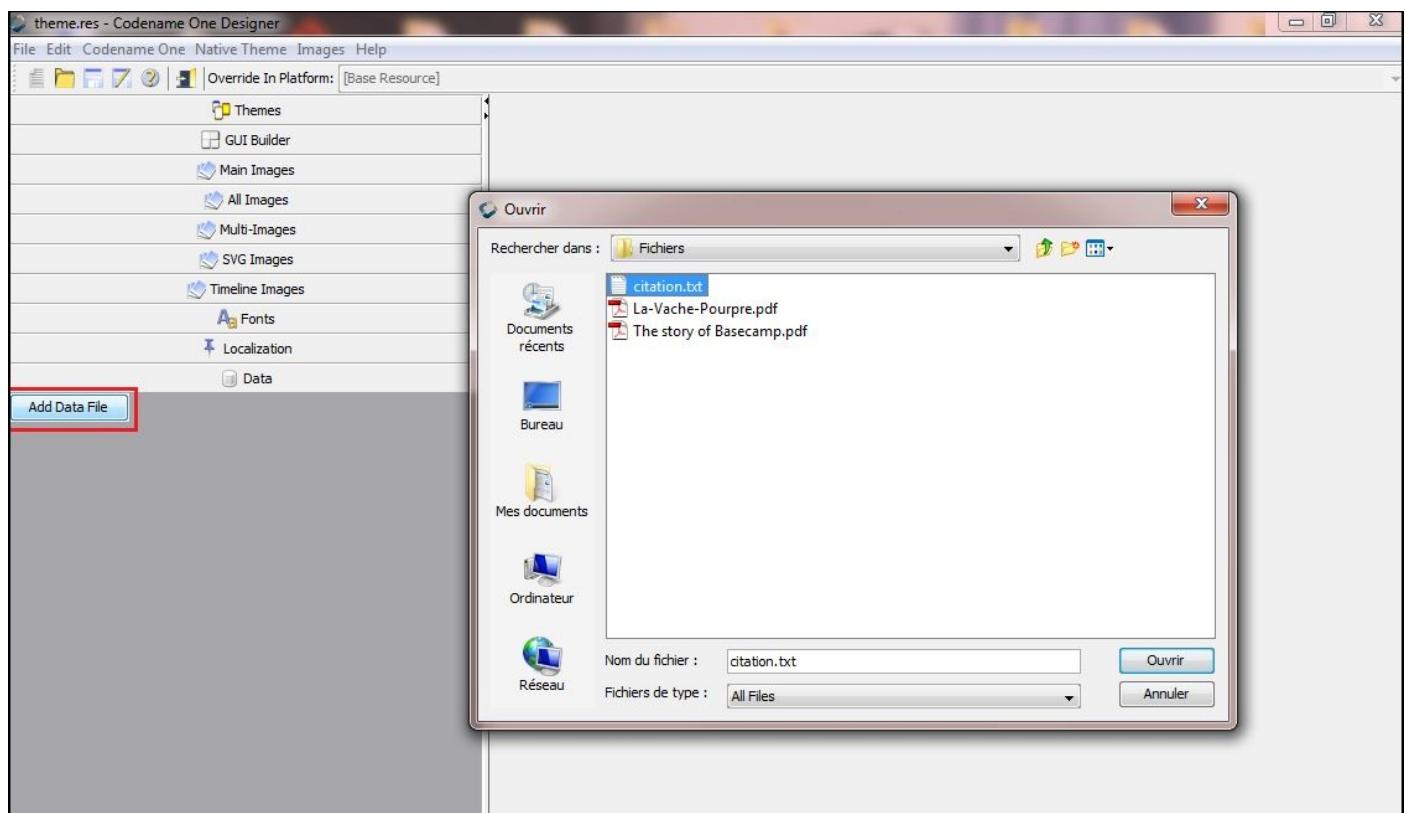
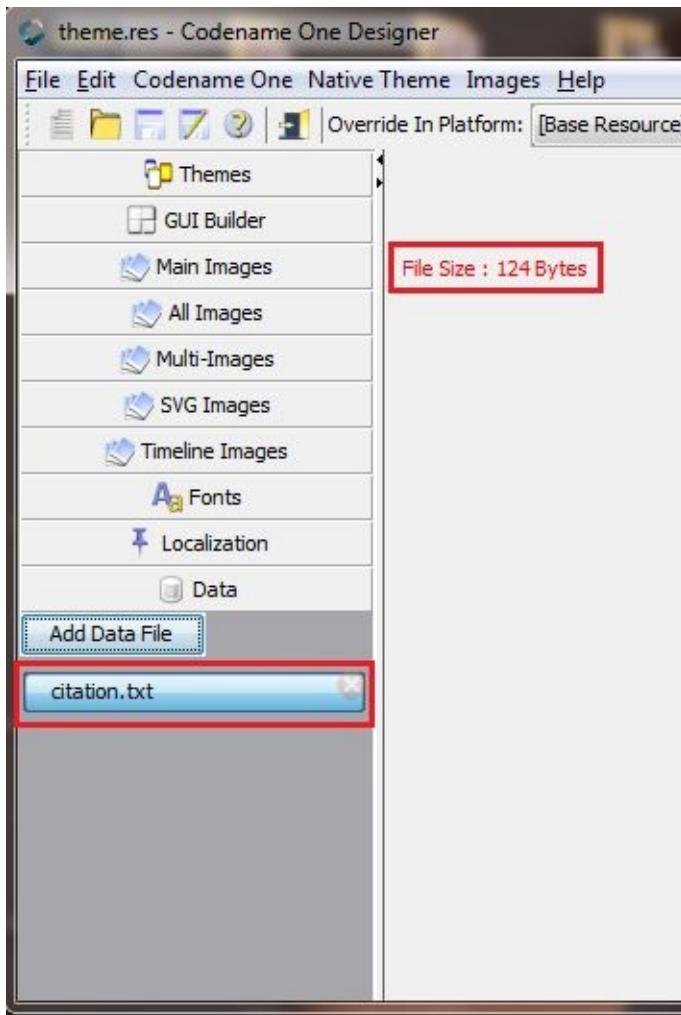


Figure 6.10 : Aperçu du fichier ajouté dans le Designer



De retour dans votre code, vous pouvez récupérer le contenu de ce fichier sous forme d'InputStream et en faire ce que vous voulez. Le fichier peut être de n'importe quel format. Voici un exemple qui récupère depuis le fichier de ressources le contenu d'un fichier texte nommé `citation.txt` et qui l'affiche dans une boîte de dialogue (voir [Figure 6.11](#)).

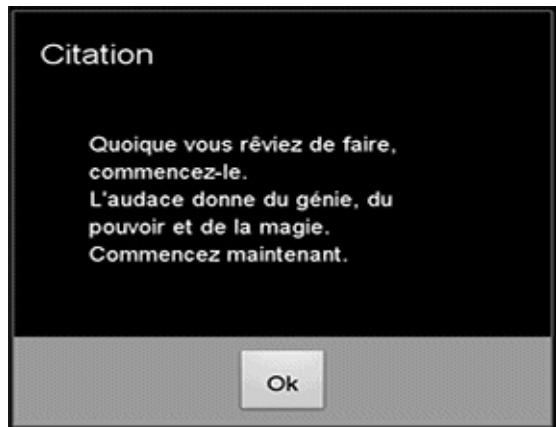
```
private Resources theme;
//...
InputStream stream=theme.getData("citation.txt");
```

❶ **try{** String contenu=Util.readToString(stream); ❷ Dialog.show("Citation", contenu, "Ok", null); ❸ } **catch** (IOException e) { Dialog.show("Erreur", e.getMessage(), "Ok", null); }

Récupération du contenu du fichier `citation.txt` sous forme d'InputStream avec la ❶ méthode `getData()` de la classe `Resources`. `getData()` prend en paramètre le nom du fichier à lire et son extension.

- ❷ `readToString()` de la classe `Util` permet de retourner sous forme de chaîne de caractères un flux `InputStream`. Ici, nous le récupérons dans la variable `contenu`.
- ❸ Affichage de la valeur de la variable `contenu` dans une boîte de dialogue.

Figure 6.11 : Contenu du fichier texte citation.txt



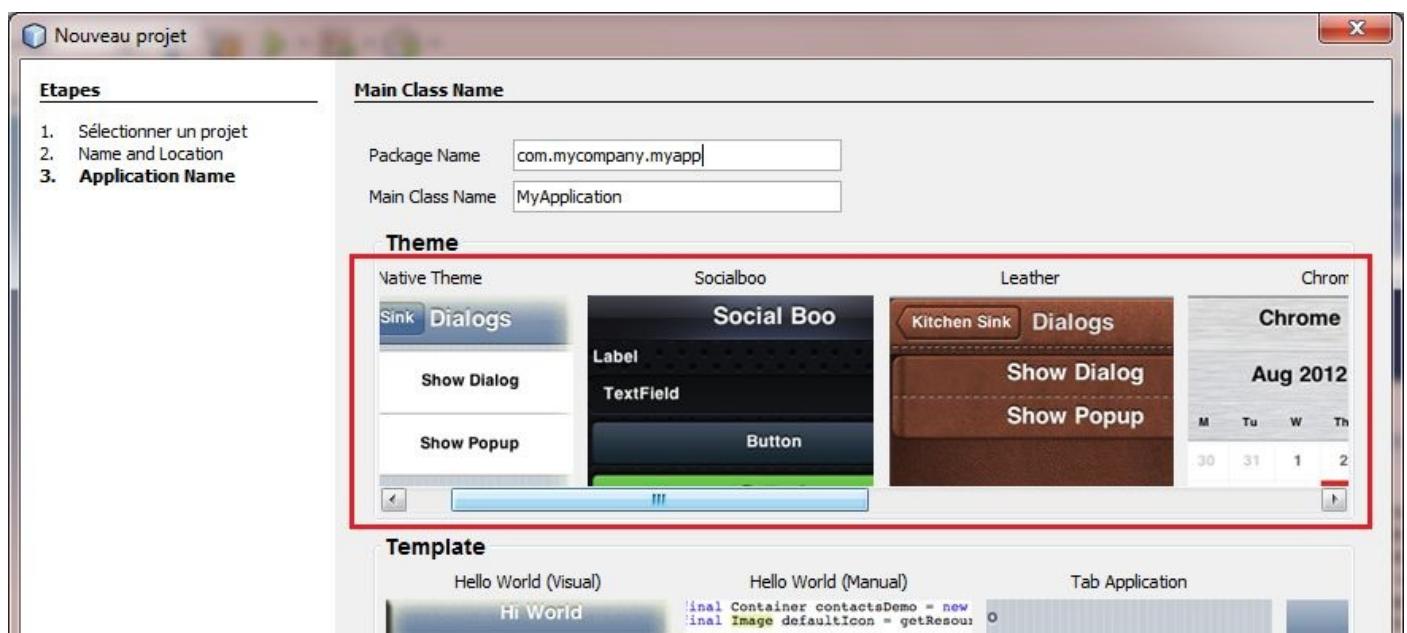
4. Les thèmes

Les thèmes sont des habillages visuels permettant de définir l'apparence des composants graphiques qui composent une interface. L'apparence est gérée par la classe Style et chaque composant a quatre styles d'objets : un objet Style pour l'état par défaut quand le composant n'est pas sélectionné (Unselected), un deuxième pour l'état du composant quand il est sélectionné (Selected), un troisième quand on appuie sur le composant (Pressed) et un quatrième quand le composant est désactivé (Disabled). Le style du troisième état ne concerne pas tous les composants, mais seulement ceux d'entre eux qui peuvent être appuyés comme un Button, un CheckBox, un RadioButton par exemple.

4.1. Utilisation des thèmes existants

En plus du [thème natif](#) que nous avons utilisé dans tous les exemples jusqu'à ce chapitre, Codename One Designer propose d'autres thèmes visuellement avancés et plus jolis. À la différence du thème natif qui change et s'adapte à chaque plateforme, les autres thèmes disponibles ne changeront pas et donneront la même apparence à l'application qui les utilise quelle que soit la plateforme sur laquelle elle s'exécutera. Il y a deux manières d'utiliser les thèmes prédéfinis fournis avec Codename One. La première est de choisir le thème voulu dans la zone theme pendant la création d'un nouveau projet (voir l'encadré sur la [Figure 6.12](#)).

Figure 6.12 : Choix du thème à utiliser



La deuxième manière est de changer le thème qui est déjà en cours d'utilisation dans votre application par un autre à partir du Designer. Ouvrez le fichier de ressources de votre projet avec le Designer et sous l'onglet Themes, supprimez le thème qui s'y trouve. Cliquez ensuite sur le bouton Add A New Theme ([Figure 6.13](#)) pour afficher la fenêtre de la [Figure 6.14](#). Sur cette fenêtre, donnez un nom (ou laissez le nom par défaut) au thème

dans le champ Name et dans la liste déroulante du champ Template, choisissez le thème que vous voulez. Validez et enregistrez les modifications.

Figure 6.13 : Ajout d'un nouveau thème (étape 1)

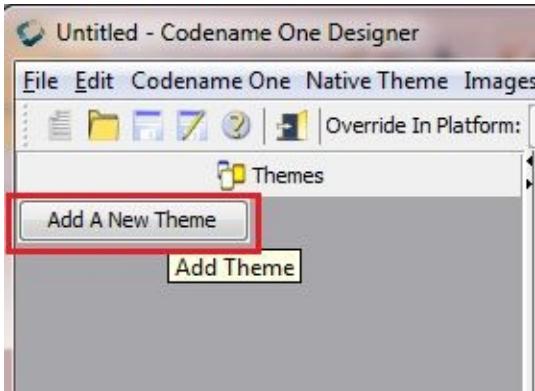
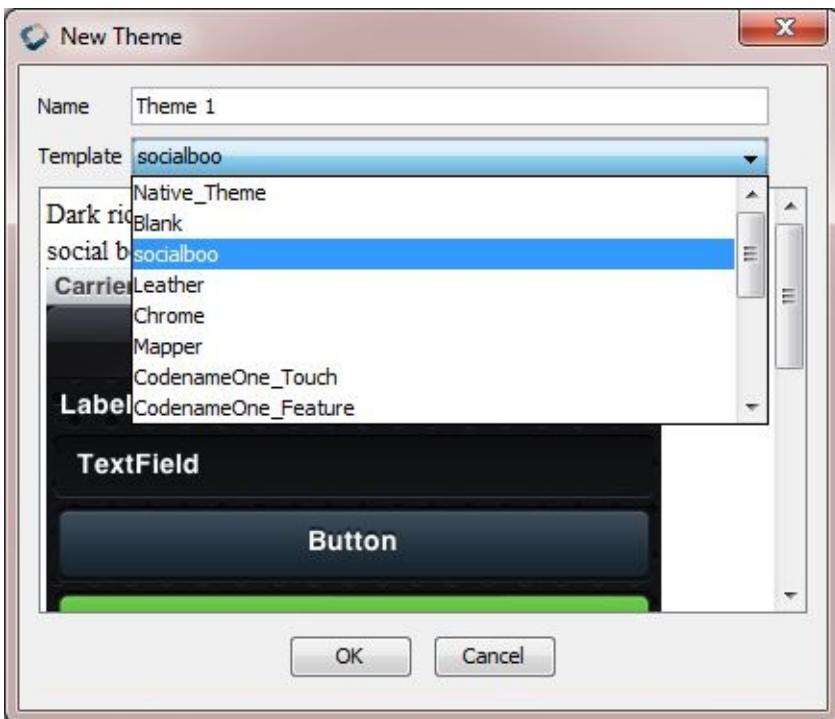


Figure 6.14 : Ajout d'un nouveau thème (étape 2)



Voici deux captures d'une application écrite en Codename One et qui utilise deux des thèmes prédéfinis.

Figure 6.15 : Application utilisant le thème Leather



Figure 6.16 : Même application utilisant le thème Chrome



Une fois qu'un thème est ajouté au Designer, il faut l'appeler depuis le code. C'est ce code qui est ajouté par défaut dans la méthode `init(Object)` à chaque fois que vous créez un nouveau projet. Comme promis au chapitre [Démarrage](#) dans le commentaire du [Hello Codename One](#), voici l'explication de ce code :

```
Resources theme=Resources.openLayered("/theme");
```

❶ `UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()[0]));` ❷

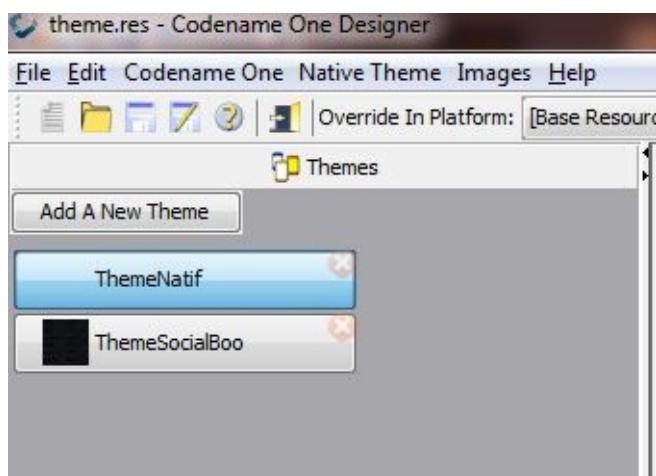
❶ Ouverture du fichier de ressources `theme.res`. L'ajout de l'extension `.res` est facultatif.

Chargement du thème et application du thème aux interfaces de l'application. La méthode `getThemeResourceNames()` retourne dans un tableau de chaînes de caractères le nom de tous les thèmes à l'intérieur du fichier de ressources. Ici, nous choisissons le premier thème de la liste d'où l'indice 0 placé entre les crochets devant la méthode `getThemeResourceNames()`.

❷ La méthode `getTheme()` permet de récupérer un thème. Elle prend en paramètre le nom du thème que vous pouvez écrire directement ou que vous pouvez récupérer comme nous venons de le faire avec `getThemeResourceNames()`. La méthode `setThemeProps()` de `UIManager` permet enfin d'appliquer le thème chargé aux interfaces de l'application.

Même s'il est possible d'avoir plusieurs thèmes dans un même fichier de ressources, on ne peut en utiliser qu'un seul à la fois. En revanche, on conserve la possibilité d'en changer à la volée même quand l'application est en pleine exécution. Voici un exemple d'une interface avec un bouton. Un clic sur celui-ci change automatiquement le thème en un autre. Avant cela, ajoutez au moins deux thèmes à votre fichier de ressources comme sur la [Figure 6.17](#).

Figure 6.17 : Ajout des thèmes Natif et SocialBoo



Le code nous donne ceci :

Exemple 6.1 : Changement de thème en pleine exécution d'une application

```

//...
private Resources theme;
//...
public void init(Object context) {
    try {
        theme=Resources.openLayered("/theme");
        UIManager.getInstance().setThemeProps(
            theme.getTheme("ThemeNatif"));
    } catch( IOException e){ e.printStackTrace(); } } public void start() { if(current!=null){
        current.show(); return; } Form f=new Form("Thèmes"); f.setLayout(new
        BoxLayout(BoxLayout.Y_AXIS)); Button b=new Button("Changer le thème");
        b.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent evt) { UIManager.getInstance().setThemeProps( theme.getTheme("ThemeSocialBoo")); }
        ② Display.getInstance().getCurrent().refreshTheme(); ③ } }); f.addComponent(b);
        f.show(); }

```

❶ Chargement et utilisation du thème natif nommé ici *ThemeNatif*.

❷ Chargement et utilisation du thème nommé *ThemeSocialBoo* après un clic sur le bouton.

❸ La méthode `refreshTheme()` appelée ici permet de rafraîchir ou d'actualiser le Form courant (et les composants placés dessus) en leur appliquant le nouveau thème chargé au point ❷. Le Form courant (celui affiché à l'écran) est récupéré ici avec `getCurrent()`.

Attention > Si vous avez d'autres Form dans votre projet, vous devez également appeler leurs méthodes `refreshTheme()` pour que le changement de thème les affecte aussi. Ceci est valable seulement pour les Form déjà créés en mémoire. Les Form qui seront créés après le changement de thème n'auront pas besoin d'appeler cette méthode et adopteront automatiquement l'habillage du nouveau thème. L'appel de `refreshTheme()` doit se faire aussi sur les Dialog avant leur affichage.

Figure 6.18 : Thème natif avant le clic sur le bouton

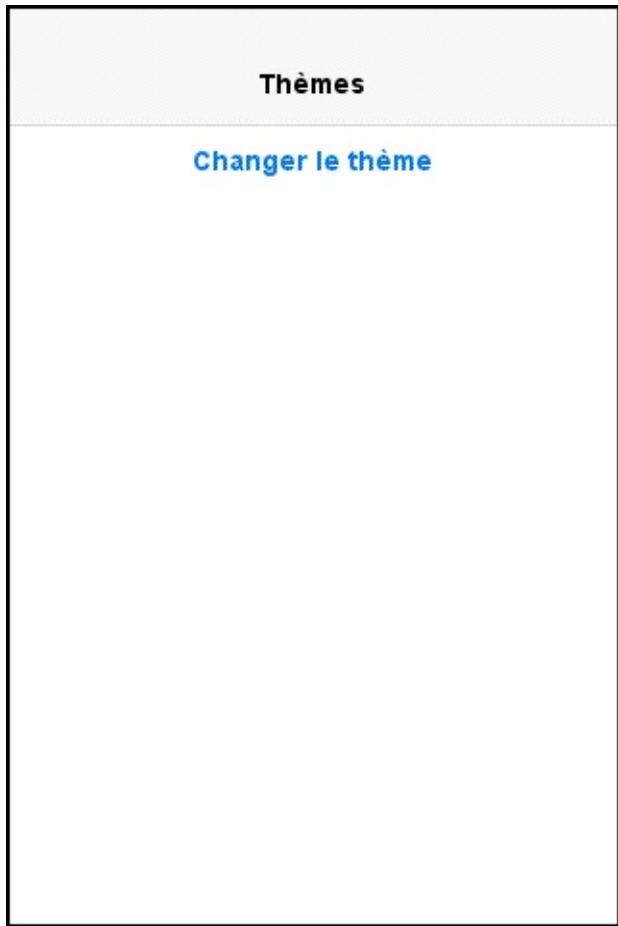
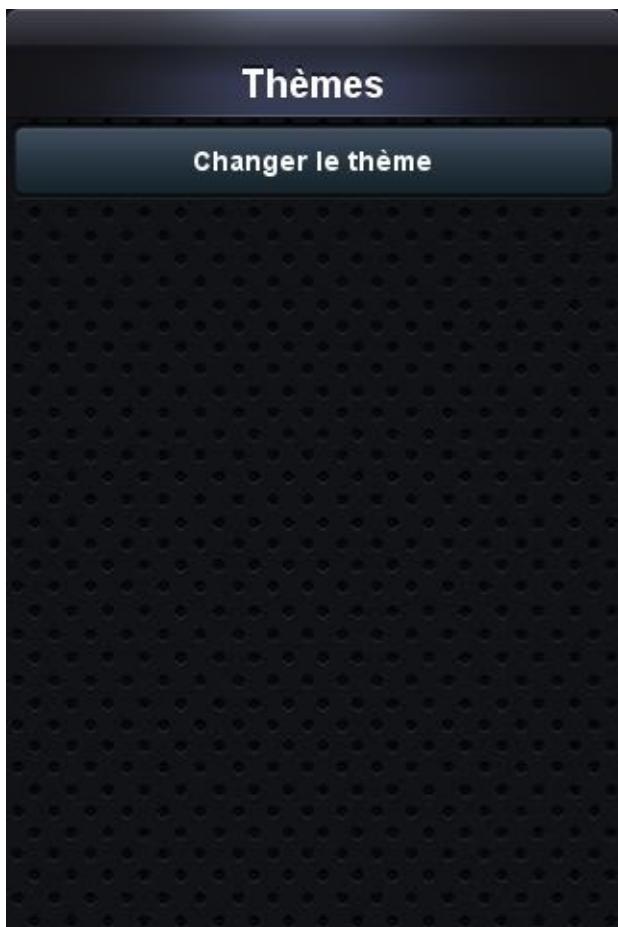


Figure 6.19 : Changement de thème (SocialBoo) après le clic sur le bouton



4.2. Concept de l'UIID

L'UIID (*User Interface IDentifier*), qu'on peut traduire en français par *identifiant d'interface utilisateur*, est un nom qu'on peut attribuer à un composant graphique. Ce nom permet d'associer des styles à un ou plusieurs composants. Même si l'UIID est distinct d'un composant graphique, ils portent généralement le même nom. Ainsi l'UIID nommé par exemple Label servira à personnaliser un composant de type Label. Un certain nombre d'UIID sont prédefinis dans Codename One et chaque composant a son UIID contenant son style visuel. Pour attribuer un UIID à un composant, utilisez la méthode `setUIID(String nomDeLUIID)`. En sus des UIID prédefinis, vous pouvez créer vos propres UIID en les nommant à votre gré et en leur attribuant les styles voulus ou en héritant des styles d'un autre UIID.

Un bon exemple de l'utilité de l'UIID est la manière dont on écrivait du texte sur plusieurs lignes avant l'apparition du composant SpanLabel dans Codename One. En ce temps, le seul composant qui permettait de créer du texte sur plusieurs lignes était le composant TextArea qui n'avait pas du tout l'apparence d'un Label. L'astuce utilisée était alors la suivante. On créait un TextArea (une zone de texte) contenant le texte voulu ❶. Ensuite, on attribuait l'UIID Label au TextArea pour lui donner exactement l'apparence visuelle d'un Label ❷. Pour finir, on désactivait l'édition du TextArea ❸ parce qu'un Label n'est pas destiné à être édité comme c'est le cas d'un TextArea. Voici le code correspondant à cet exemple d'utilisation de l'UIID :

Exemple 6.2 : Utilisation de l'UIID sur un composant

```
Form f=new Form();
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

TextArea aPropos=new TextArea("Codename One est un framework de
développement d'applications mobiles multiplateformes qui...");

❶ aPropos.setUIID("Label");
❷ aPropos.setEditable(false);
❸ f.addComponent(aPropos);
f.show();
```

Figure 6.20 : Un TextArea normal contenant du texte

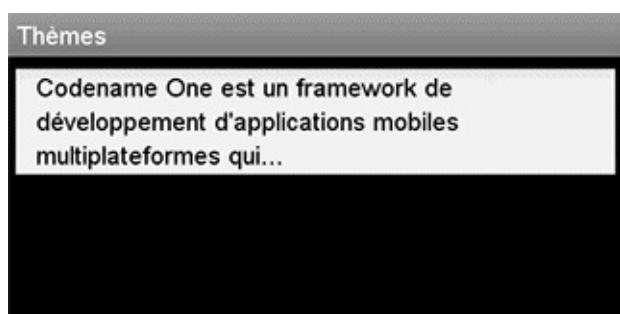
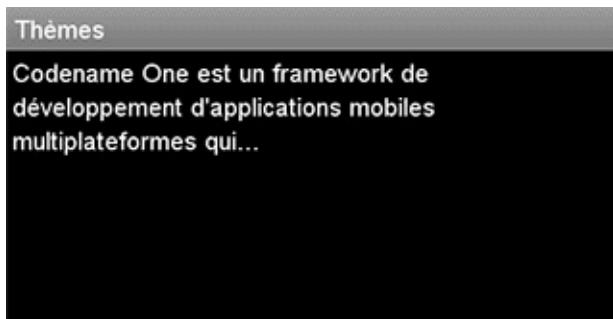
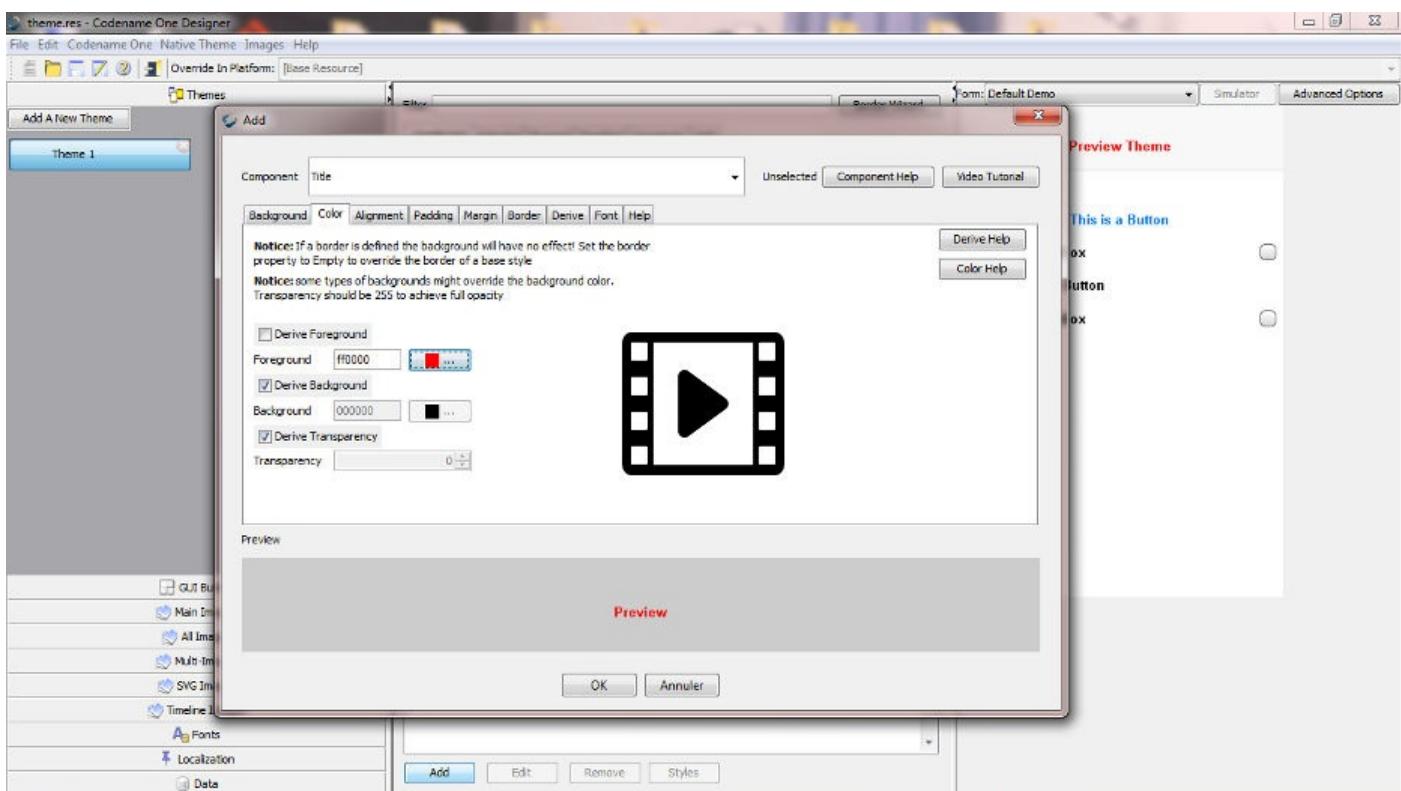


Figure 6.21 : Un TextArea avec l'UIID Label



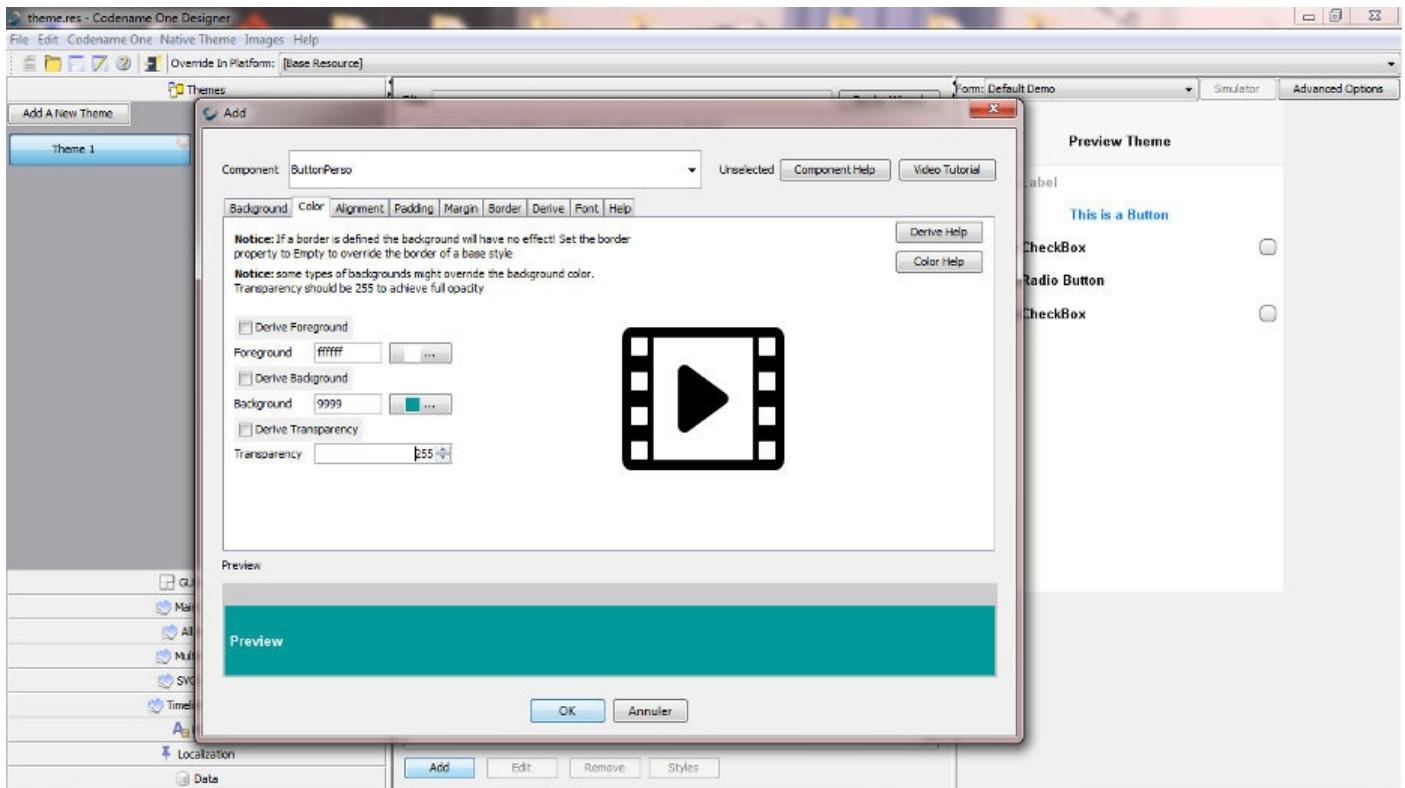
L'attribution ou la personnalisation des styles d'un UIID prédefini ou crée par vous-même peut se faire avec du code, mais le plus simple est de le faire avec le Designer, qui vous permettra de gagner un temps considérable. La vidéo suivante ([Figure 6.22](#)) vous montre comment personnaliser un UIID et la manière dont il affecte un composant du même nom.

Figure 6.22 : Création d'un UIID prédefini et personnalisation de son style (vidéo)



Voici une autre vidéo ([Figure 6.23](#)) qui montre comment créer votre UIID en faisant hériter certains de ses styles d'un UIID existant et l'appliquer au composant de votre choix.

Figure 6.23 : Création d'un UIID personnalisé (vidéo)



4.3. Bases de la création d'un thème

Maintenant que vous savez à quoi sert un UIID, nous allons l'utiliser pour créer un mini-thème à partir de zéro. Dans notre mini-thème, nous personnaliserez uniquement certains composants que voici :

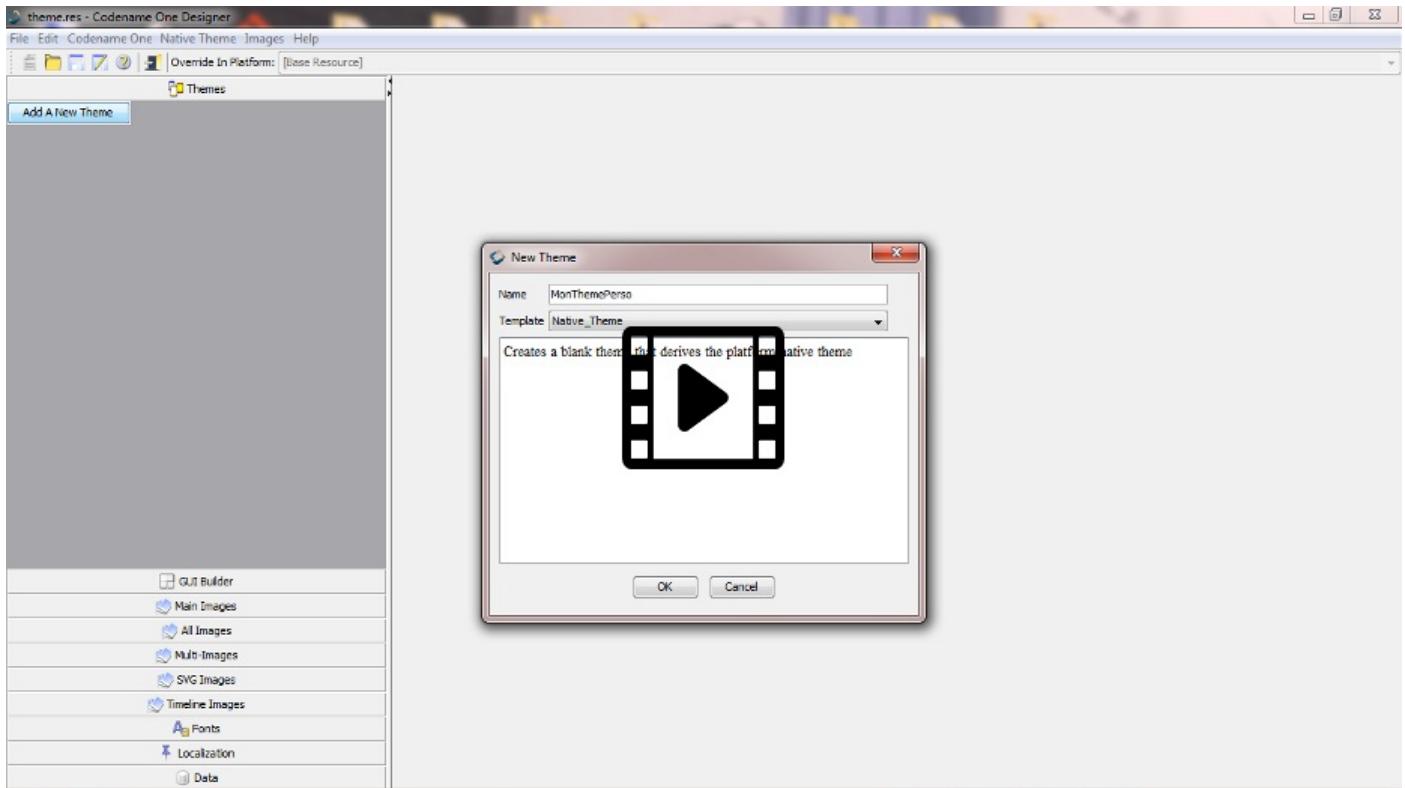
- Label ;
- Button ;
- TextField ;
- zone de titre de la fenêtre ;
- zone de contenu de la fenêtre.

Étant donné que nous allons baser notre thème sur le thème natif, les composants que nous ne personnaliseras pas garderont leur apparence visuelle native, celle-ci dépendant de la plateforme où s'exécutera l'application. La création de notre thème sera constituée des étapes suivantes :

- création d'un thème natif ;
- personnalisation des UIID des trois composants et des deux zones précédemment cités.

La vidéo de la [Figure 6.24](#) montre le processus complet.

Figure 6.24 : Créeation d'un mini-thème (vidéo)



4.4. Les constantes de thèmes

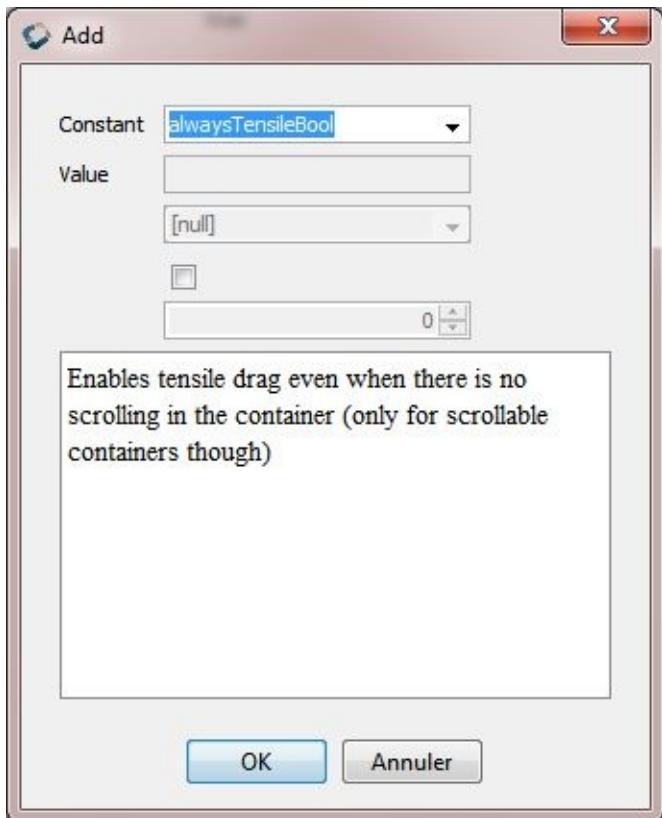
Les constantes de thèmes permettent d'attribuer des comportements généraux à des composants dans Codename One. Elles sont très pratiques en ce sens qu'*il suffit de les utiliser pour que l'action voulue soit répercutée sur toute l'application*. À l'instant où ces lignes sont écrites, il y a au total 140 constantes et ce nombre continue d'augmenter régulièrement. Compte tenu de ce nombre élevé et de la rapidité avec laquelle cette liste s'élargit, nous ne vous la fournirons pas dans cet ouvrage. Nous vous invitons à la consulter directement au [chapitre Advanced Theming](#) du manuel en ligne de Codename One.

Pour utiliser une constante de thème, vous devez suivre la procédure suivante. Ouvrez le thème de votre application dans le Designer. Allez ensuite sous l'onglet Constants puis cliquez sur le bouton Add en bas de la fenêtre ([Figure 6.25](#)). Dans la fenêtre qui s'affichera ([Figure 6.26](#)), choisissez la constante de thème à utiliser dans la zone Constant puis attribuez-lui une valeur dans la zone Value. La valeur d'une constante de thème peut être une chaîne de caractères, un booléen, une image ou encore un entier.

Figure 6.25 : Ajout d'une constante de thème (étape 1)



Figure 6.26 : Ajout d'une constante de thème (étape 2)



Un exemple d'utilisation des constantes de thèmes est le suivant. Nous avons vu au chapitre [Les composants graphiques](#) qu'on pouvait appliquer des effets de transition à un Form ou à un Dialog. Supposons que notre application soit composée de plusieurs Form et Dialog et que nous voulions par exemple appliquer le même effet de transition à tous les Form et à tous les Dialog. Il nous faudrait alors écrire le code de transition pour chacun des nombreux Form ou Dialog de l'application. Dans ce cas, nous aurons tout intérêt à recourir aux constantes de thèmes. Pour les Form, nous pouvons utiliser la constante `formTransitionOut` ou `formTransitionIn`. Cette constante aura par exemple pour valeur le mot *fade* (pour une transition de fondue enchaîné) ou *slide* (pour une transition d'effet de carrousel). Et pour les Dialog, `dialogTransitionOut` et `dialogTransitionIn`.

Note > Vous trouverez également un exemple d'utilisation de constantes de thèmes à la [Section 6, Menu hamburger](#).

5. L'éditeur d'interfaces graphiques (GUI Builder)

Le GUI Builder est le nom de l'éditeur graphique de Codename One. Il est intégré au Designer et permet de concevoir visuellement avec un mécanisme de glisser/déposer les différentes fenêtres d'une application. Dans le cas de la création des interfaces uniquement, le GUI Builder peut être utilisé seul sans l'environnement de développement qui servira plus tard à écrire la logique de code de l'application.

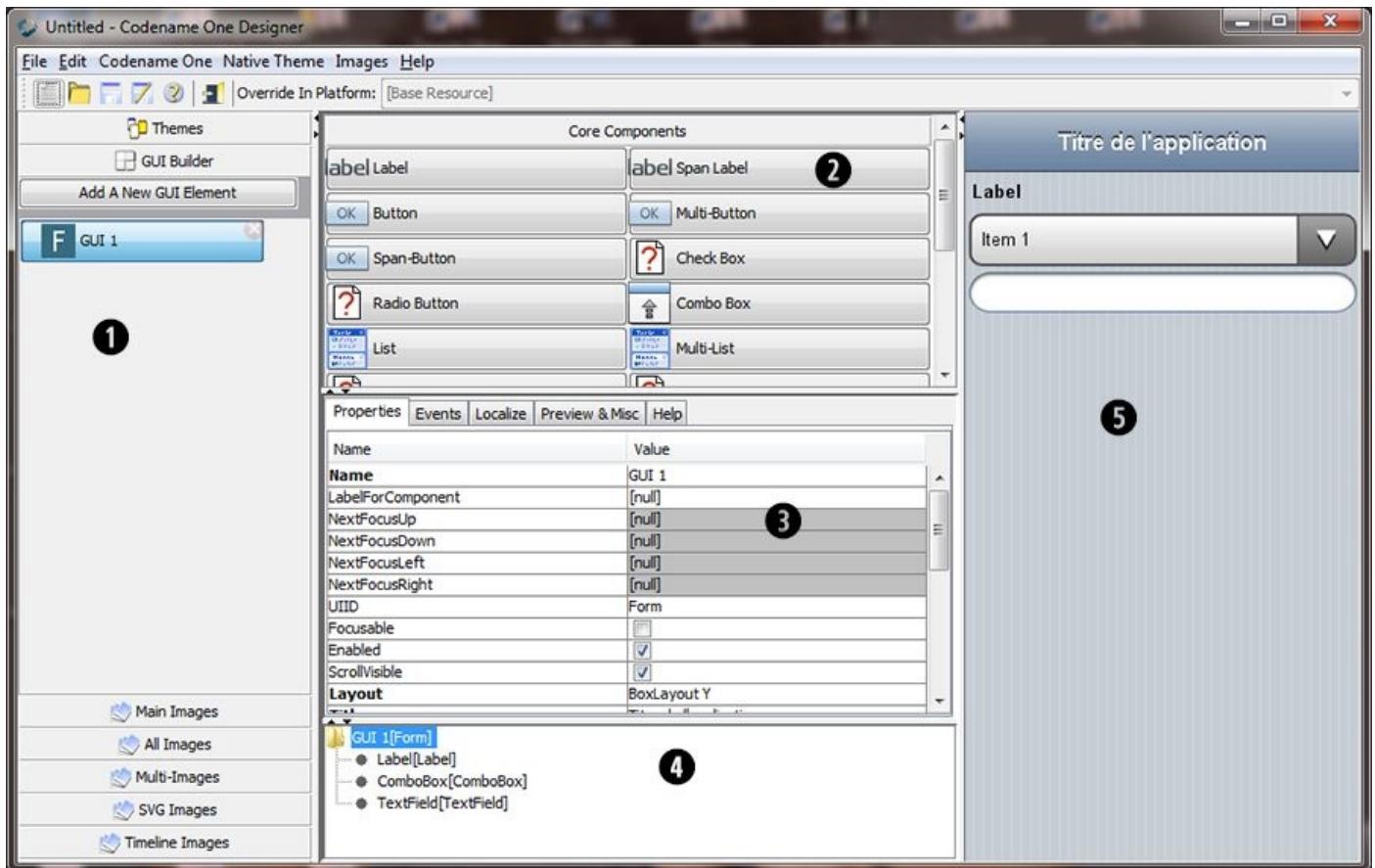
Il simplifie la conception d'interfaces et permet de gagner du temps. Son emploi, cependant, ne présente pas que des avantages. En exemple, la communication entre le GUI Builder et le code n'est pas intuitive. De plus, les codes qui interagissent avec les interfaces dessinées se retrouvent dans un fichier unique (même classe) quel que soit le nombre de fenêtres créées. Un autre inconvénient est que les applications conçues avec le GUI Builder sont plus lourdes en taille et consomment aussi plus en mémoire que celles écrites manuellement avec du code uniquement. Cela peut se justifier par le fait que les interfaces dessinées grossissent la taille du fichier de ressources qui occupe lui-même la plus grande taille dans l'exécutable final de l'application.

Astuce > Compte tenu des inconvénients précités, je conseillerais de ne pas utiliser le GUI Builder pour de gros et moyens projets. Utilisez-le seulement pour de petits projets ou pour faire des prototypes rapidement. Pour les projets de grande taille, optez pour l'écriture des interfaces au code.

5.1. Présentation de l'interface et fonctionnement du GUI Builder

Le GUI Builder est constitué de cinq zones redimensionnables que voici :

Figure 6.27 : Interface du GUI Builder



- ❶ Liste de toutes les interfaces (fenêtres) créées.
- ❷ Liste des composants graphiques à glisser/déposer vers la zone ❸.
- ❸ Liste des propriétés du composant sélectionné sur l'interface.
- ❹ Hiérarchie des composants utilisés sur l'interface.
- ❺ Vue principale contenant le rendu de l'interface à éditer.

Attention > Chaque fois qu'un enregistrement est effectué dans le fichier de ressources, le GUI Builder génère automatiquement une classe nommée `StateMachineBase`. Il est totalement inutile d'ajouter du code dans le fichier de cette classe parce qu'il est toujours régénéré, les ajouts manuels y sont donc perdus. Les ajouts de code doivent être faits dans la classe `StateMachine`. Cette dernière hérite de `StateMachineBase` qui hérite elle-même de la classe `UIBuilder`.

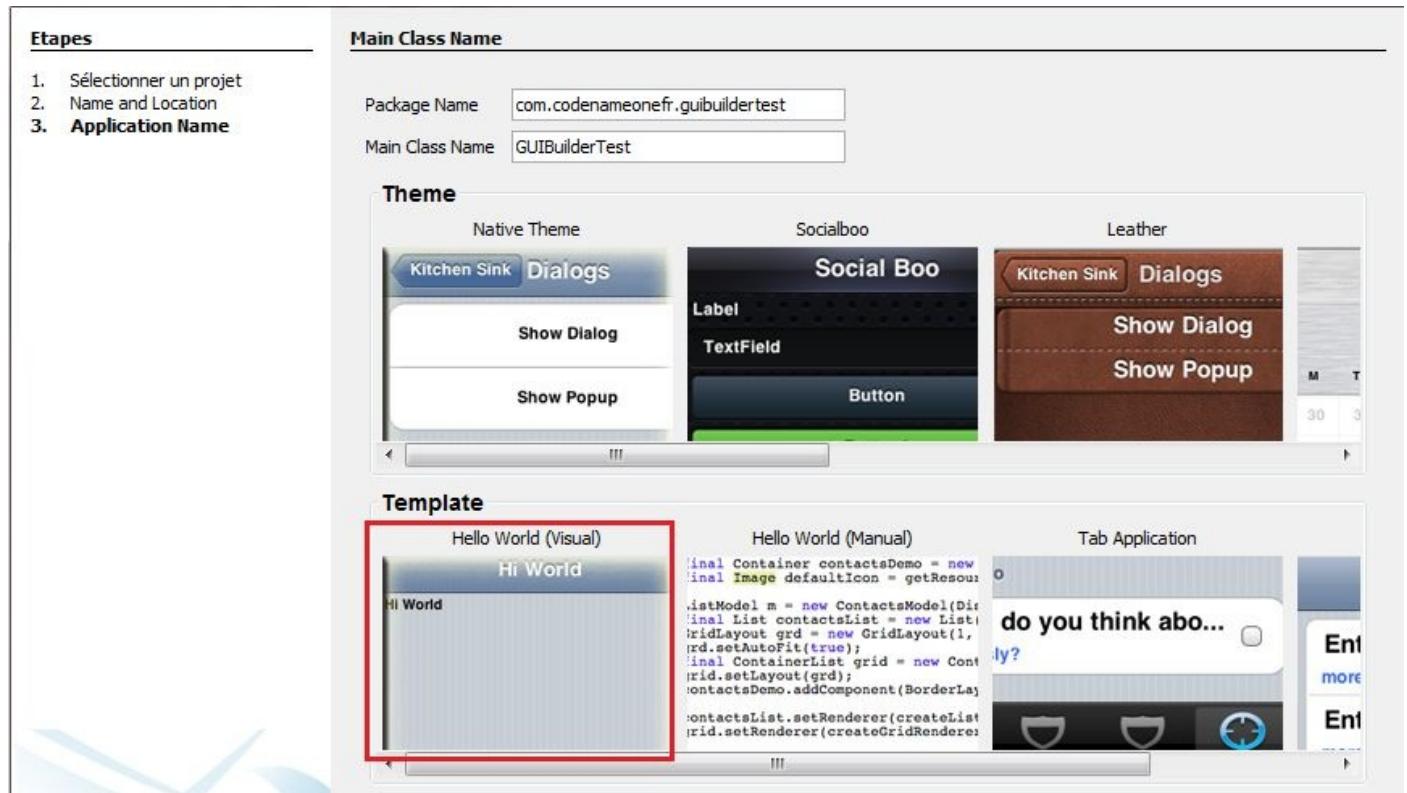
Dans la classe générée `StateMachineBase`, on trouve des méthodes qui permettent d'accéder aux composants déposés sur les interfaces et des fonctions callback qui sont appelées automatiquement quand les événements correspondant à ces fonctions sont déclenchés. Les méthodes permettant d'accéder aux composants des interfaces se présentent sous la forme `findXXX()` où `XXX` correspond au nom du composant à utiliser. En exemple, la méthode `findNom()` permet d'accéder à un composant nommé `Nom` dans le GUI Builder. Une version surchargée de cette méthode est disponible et permet de

spécifier en paramètre le composant parent du composant à sélectionner.

5.2. Crédation d'interfaces graphiques

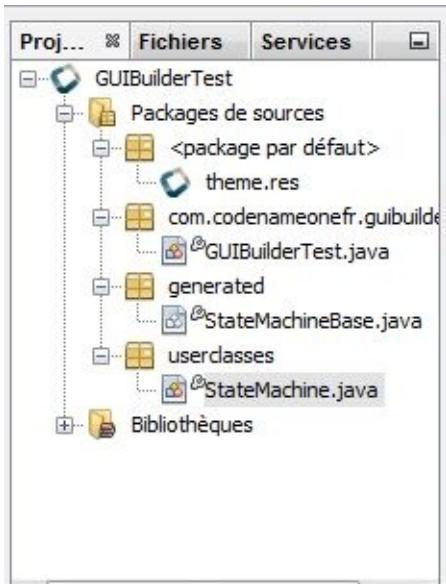
Pour créer un projet fondé sur l'architecture du GUI Builder, il faut choisir le template Hello World (Visual) (voir [Figure 6.28](#)) au lieu du [Hello World \(Manual\)](#) tel que nous l'avons fait jusqu'ici.

Figure 6.28 : Crédation d'un projet basé sur le GUI Builder



La structure des fichiers dans le dossier SRC diffère un peu de celle que nous avons déjà rencontrée et ressemble à celle de la [Figure 6.29](#).

Figure 6.29 : Structure d'un projet utilisant le GUI Builder



Le fichier `GUIBuilderTest.java` contient le code du [cycle d'exécution de l'application](#) et le fichier `StateMachine.java` les codes qui feront fonctionner l'application.

La création d'interfaces avec le GUI Builder étant assez intuitive, nous nous contenterons de vous montrer à l'aide d'une vidéo un exemple de création de deux fenêtres (Form) contenant des formulaires à remplir.

Pour créer une fenêtre, lancez le designer en ouvrant le fichier de ressources de notre projet (`theme.res`). Sous l'onglet GUI Builder du designer, cliquez sur Add A New GUI Element pour afficher une fenêtre. Sur cette fenêtre, donnez un nom à votre Form dans le champ Name et validez (voir la [Figure 6.30](#)). Vous pouvez maintenant commencer à ajouter des éléments sur le Form créé comme illustré dans la vidéo de la [Figure 6.31](#).

Figure 6.30 : Étapes de création d'une fenêtre

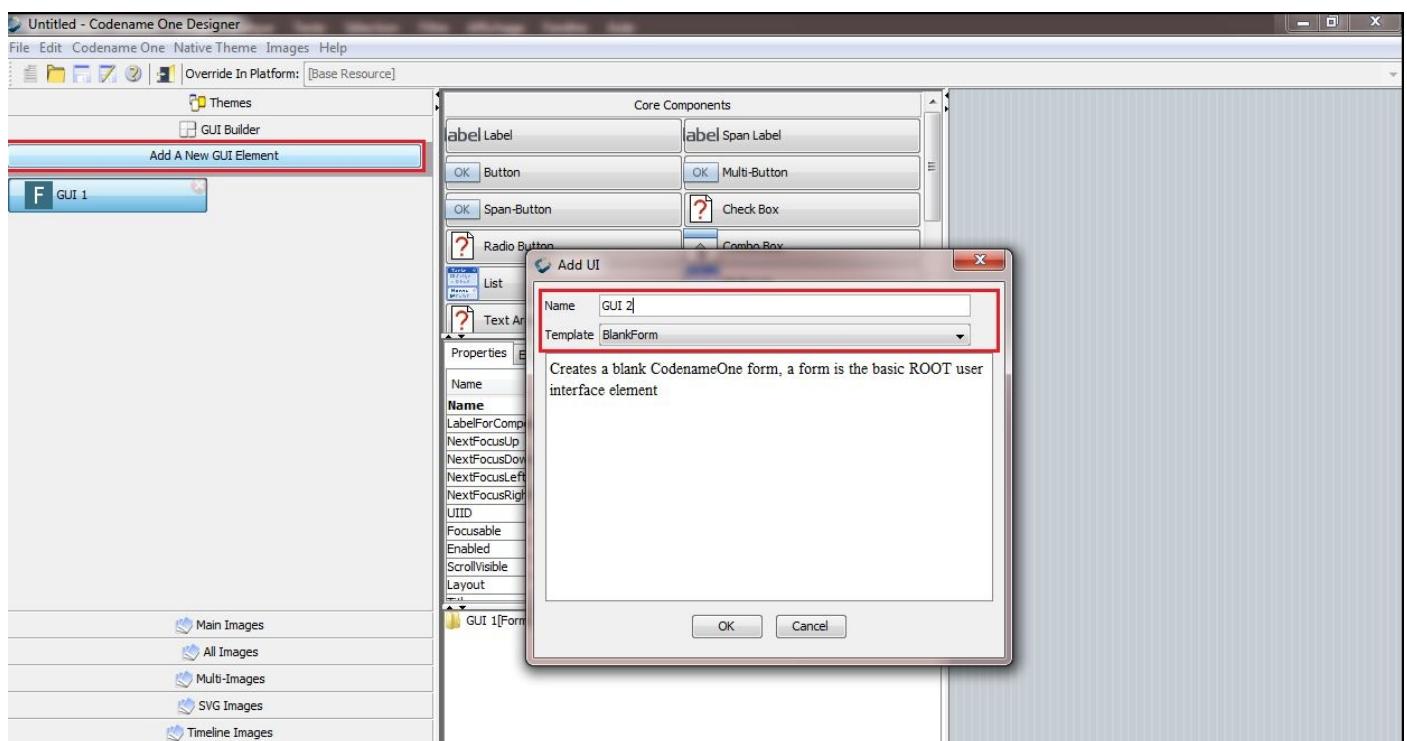
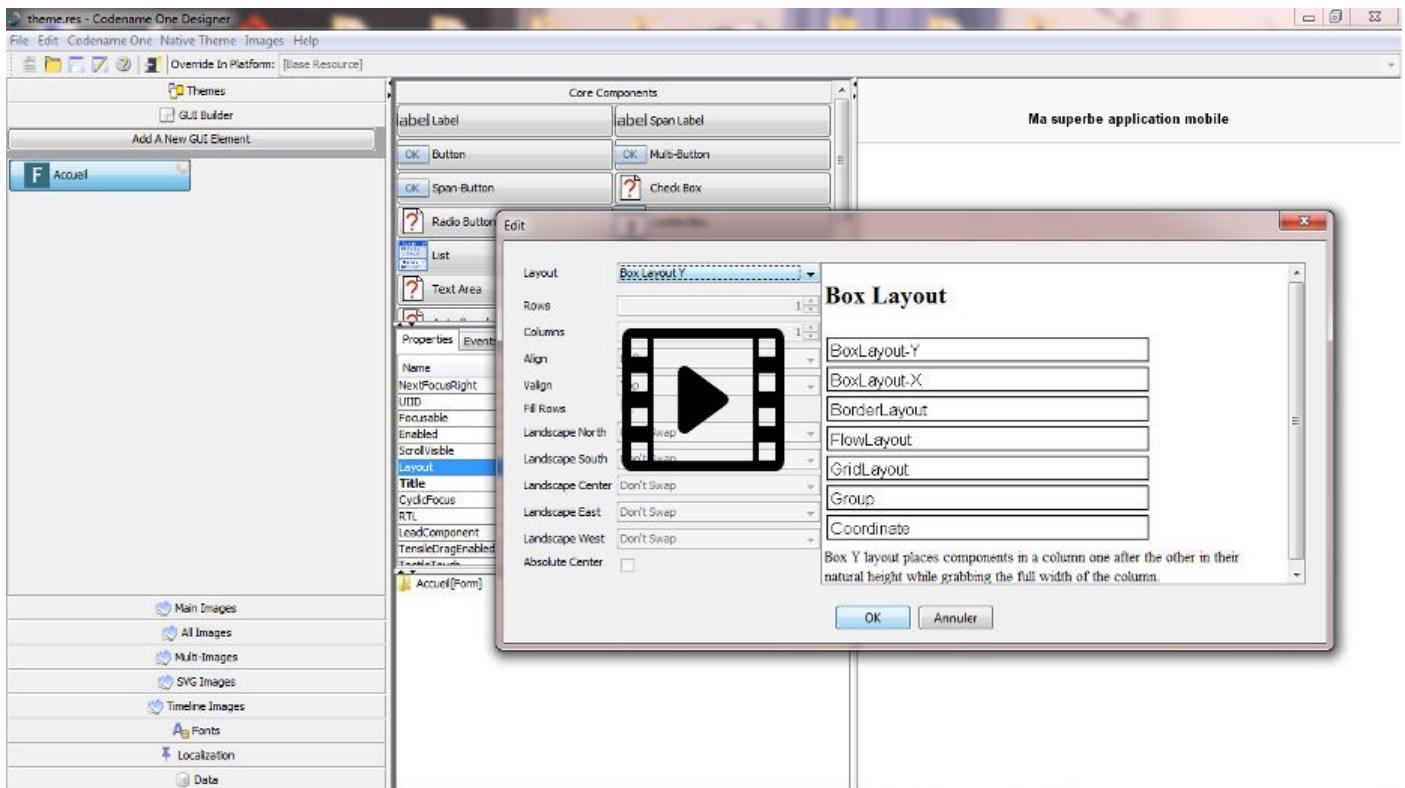


Figure 6.31 : Création d'une interface avec le GUI Builder (vidéo)

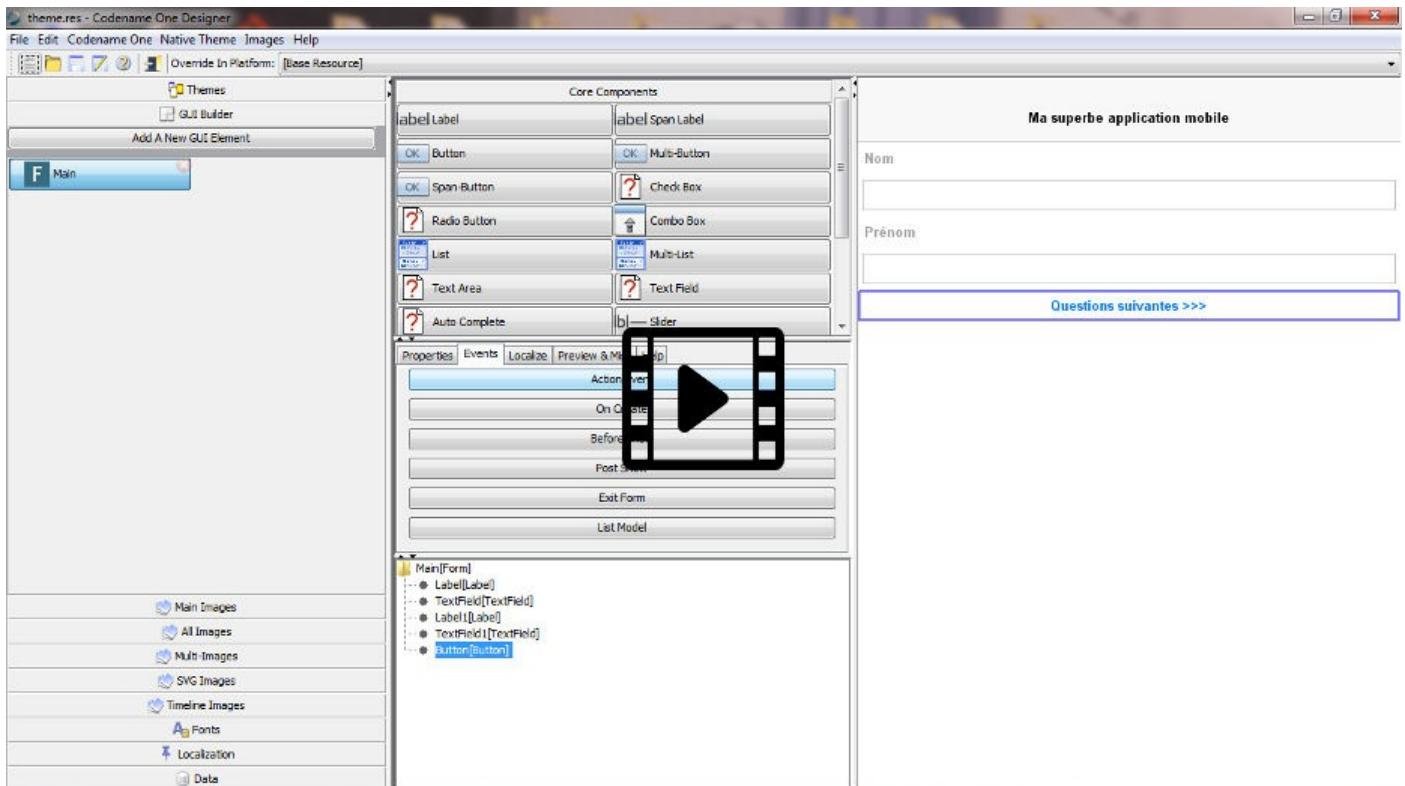


5.3. Gestion des événements

Les deux fenêtres étant créées, nous allons voir maintenant avec la vidéo suivante comment gérer les événements.

À ce niveau, nous allons utiliser les deux fenêtres créées à la section précédente. Un clic sur le bouton présent sur la première fenêtre affichera la seconde. Pour cela, nous allons sélectionner le bouton de la première fenêtre et sous l'onglet Events (à côté de l'onglet Properties), nous cliquerons sur le bouton ActionEvent pour générer une méthode dans l'éditeur de code. C'est cette méthode que nous allons implémenter. Voir la vidéo pour l'illustration.

Figure 6.32 : Gestion des événements avec le GUI Builder (vidéo)



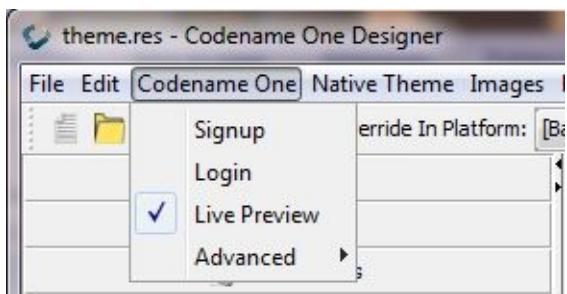
5.4. Codename One LIVE et prévisualisation en live (Live Preview)

Attention > Cette fonctionnalité est réservée aux utilisateurs PRO ou supérieur.

CodenameOne LIVE est une application mobile (pour iOS et Android) qui permet de voir en instantané sur son smartphone ou tablette l'aperçu des interfaces que vous dessinez avec le GUI Builder. Pour bénéficier de cette fonctionnalité, ouvrez le Designer puis cochez l'entrée Live Preview du menu Codename One (voir Figure 6.33).

L'application Codename One LIVE permet aussi d'accéder directement à votre espace membre où sont stockés les exécutables de vos applications. Cela permet de les installer automatiquement sur votre appareil sans devoir les télécharger d'abord sur un ordinateur puis de les transférer sur l'appareil avant de les installer.

Figure 6.33 : Menu Live Preview



En cochant Live Preview, vous serez invité à entrer vos identifiants. Pour effectuer un test,

retournez au GUI Builder, créez ou choisissez une page (Form) et à chaque enregistrement des modifications, vous remarquerez que l'interface est chargée automatiquement sur votre smartphone ou tablette dans l'application Codename One LIVE.

Note > Les vidéos (en anglais) [Codename One LIVE!](#) et [Live Preview On Device GUI Builder](#) montrent l'utilisation de cette application avec le Codename One Designer.

6. Polices de caractères

Codename One supporte deux types de polices de caractères. Les polices systèmes et les polices de type TTF (True Type Font). Les polices systèmes sont basées sur les polices disponibles à l'intérieur de l'appareil. Elles sont présentes sur toutes les plateformes supportées par Codename One tandis que les polices TTF ne fonctionnent que sous iOS, Android et BlackBerry. Les polices systèmes étant gérées par défaut, nous ne verrons que l'intégration et l'utilisation des polices True Type.

La première chose à faire est de vous rendre sur Internet pour acquérir une police TTF qui vous plaît. Une fois que vous avez trouvé celle que vous voulez utiliser, copiez son fichier dans le dossier SRC du projet. Voyons maintenant comment l'utiliser dans votre programme via [le code](#) et [le Designer](#).

Note > Pensez à vérifier la licence de la police de caractères que vous souhaitez utiliser. Toutes ne sont pas autorisées à un usage commercial.

6.1. Utilisation des polices TTF avec du code

Pour l'exemple, nous allons appliquer à un Label une police TTF nommée *Handlee*. Comme toutes les images et autres ressources utilisées ici, vous trouverez le fichier de cette police dans les sources du livre. Une fois le fichier *Handlee.ttf* ajouté au dossier SRC, passons au code de chargement de cette police et de son application à un Label.

```
Label l=new Label("Bonjour");  
① if(Font.isTrueTypeFileSupported()){ ② Font  
font=Font.createTrueTypeFont("Handlee", "Handlee.ttf"); ③ font=font.derive(50,  
Font.STYLE_BOLD); ④ l.getUnselectedStyle().setFont(font); ⑤ }
```

① Création du Label sur lequel la police sera appliquée.

La méthode `isTrueTypeFileSupported()` permet de vérifier si la plateforme sur laquelle s'exécute l'application supporte les polices True Type Font. Elle retourne `true` si c'est le cas et `false` dans le cas contraire. Si la plateforme ne supporte pas ce type de police alors la police système sera utilisée.

Création d'un objet Font de type TTF avec la méthode `createTrueTypeFont()` qui ③ prend en premier paramètre le nom de la police et en deuxième paramètre le nom du fichier de la police et son extension. Il est important d'ajouter l'extension `.TTF`.

④ La méthode `derive()` utilisée ici nous permet de définir la taille à utiliser (en premier paramètre) et le style [GRAS, ITALIQUE ou NEUTRE] (en deuxième paramètre).

Pour finir, nous appliquons l'objet Font au Label avec la méthode `setFont()` de la

❸ classe Style. getUnselectedStyle() retourne un objet de type Style et définit le style de l'état par défaut du Label.

La [Figure 6.34](#) vous montre ce que ça donne.

Figure 6.34 : Aperçu de la police Handlee appliquée à un Label



Si vous souhaitez utiliser la police sur un nombre restreint de composants, alors il suffit de l'appliquer sur ces derniers avec setFont(). En revanche, si vous avez l'intention de l'appliquer à tous les Label qui seront créés ou encore à tous les composants qui contiennent du texte, alors il vaut mieux passer par le Designer. C'est ce que nous allons voir à la section suivante.

6.2. Utilisation des polices TTF avec le Designer

Pour appliquer une police TTF à tous les Label de l'application sans écrire de code, nous allons recourir à la notion d'[UIID](#) que nous avons vue dans le cadre de la création des thèmes.

Pour cela, ouvrez le fichier de ressources, allez sous l'onglet Themes et sélectionnez le thème. Sous l'onglet Unselected, cliquez ensuite sur le bouton Add (voir [Figure 6.35](#)). Dans la zone Component, sélectionnez Label dans la liste déroulante. Allez ensuite sous l'onglet Font, décochez Derive Font et choisissez le fichier TTF (ici Handlee.ttf) dans la liste déroulante du champ True Type (voir [Figure 6.36](#)). Le nom du fichier est disponible à cet endroit parce que le Designer l'a détecté depuis le dossier SRC de votre projet. Vous pouvez aussi choisir la taille de la police dans la zone True Type Size qui est réglée par défaut sur une taille moyenne (Medium). Validez et fermez la fenêtre. Enregistrez les modifications, exécutez un exemple d'application contenant des Label et voyez le résultat.

Figure 6.35 : Ajout d'une police True Type Font au composant Label (étape 1)

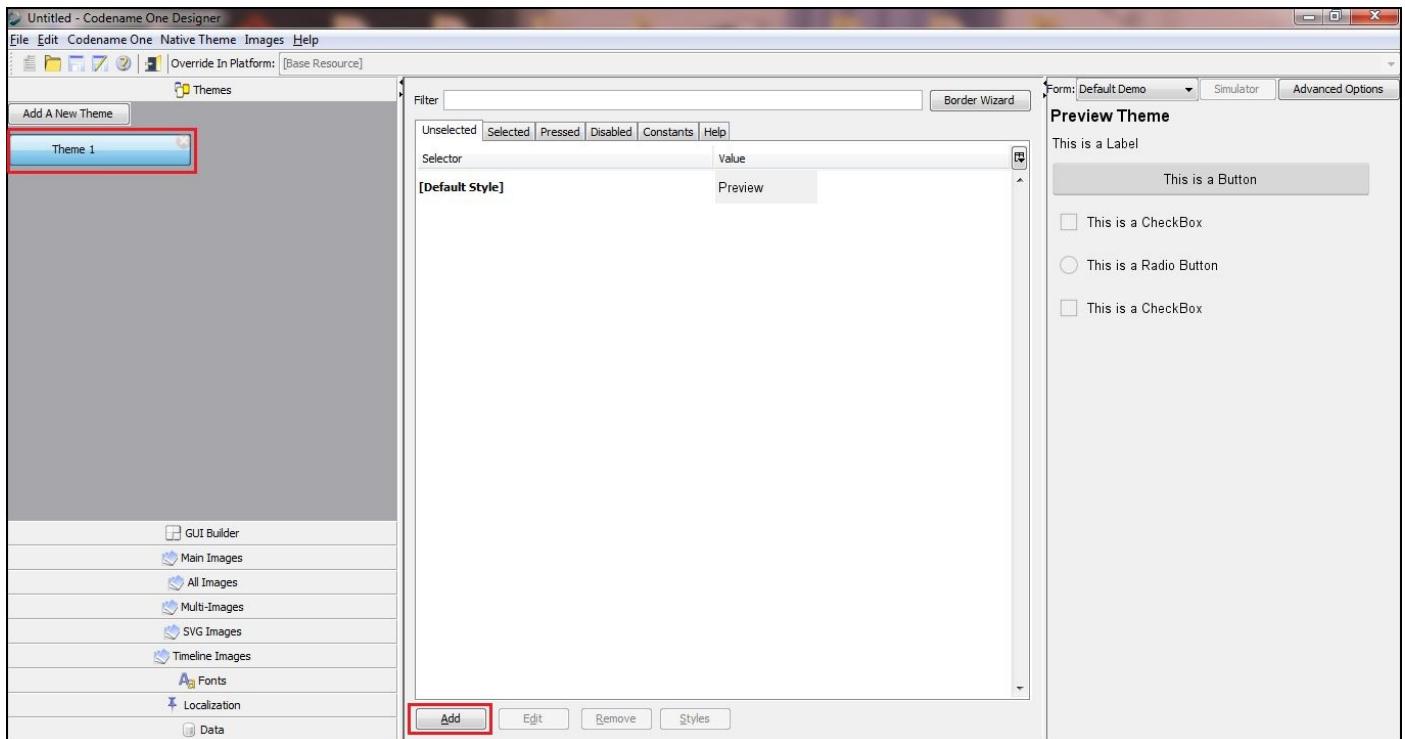
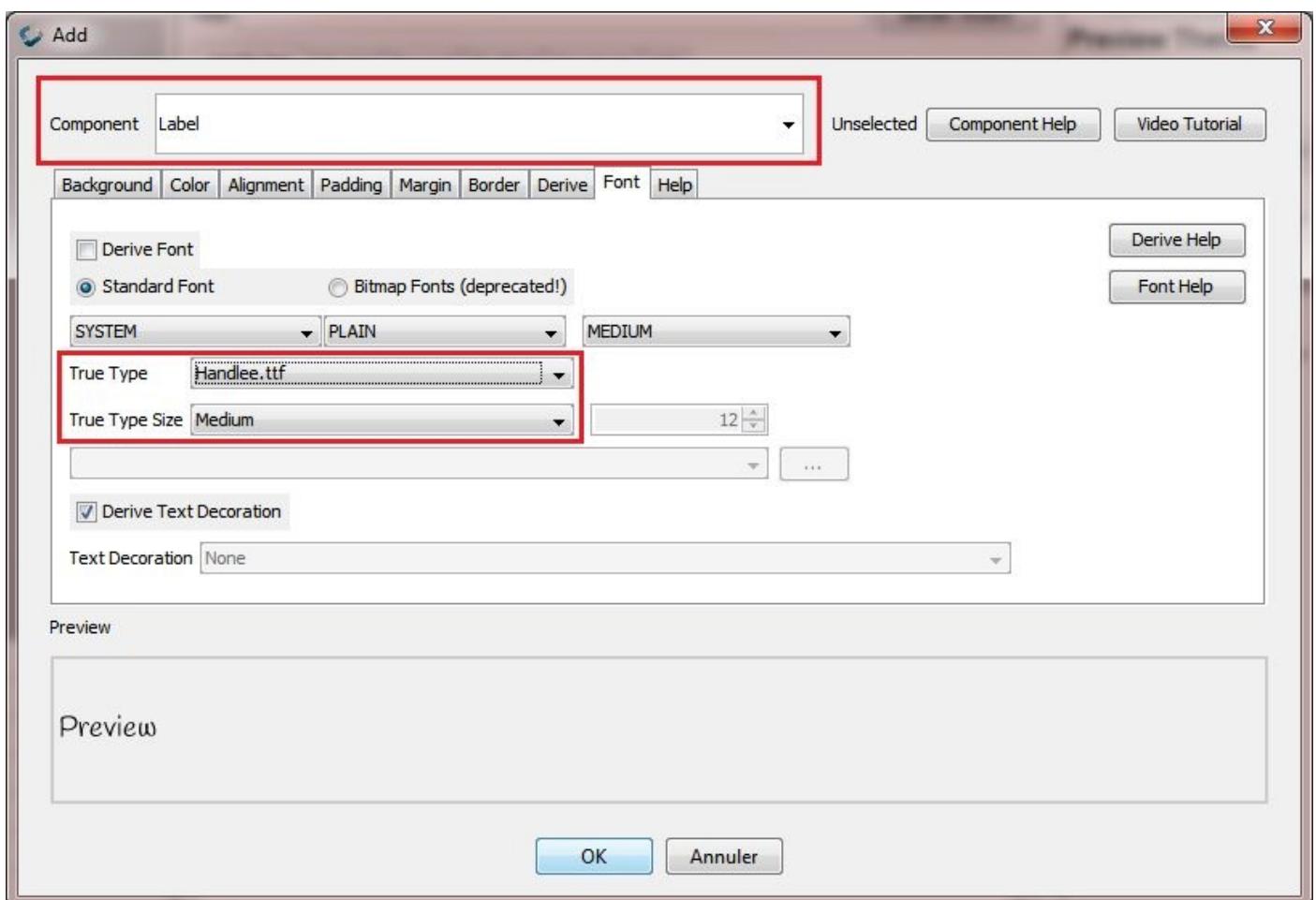
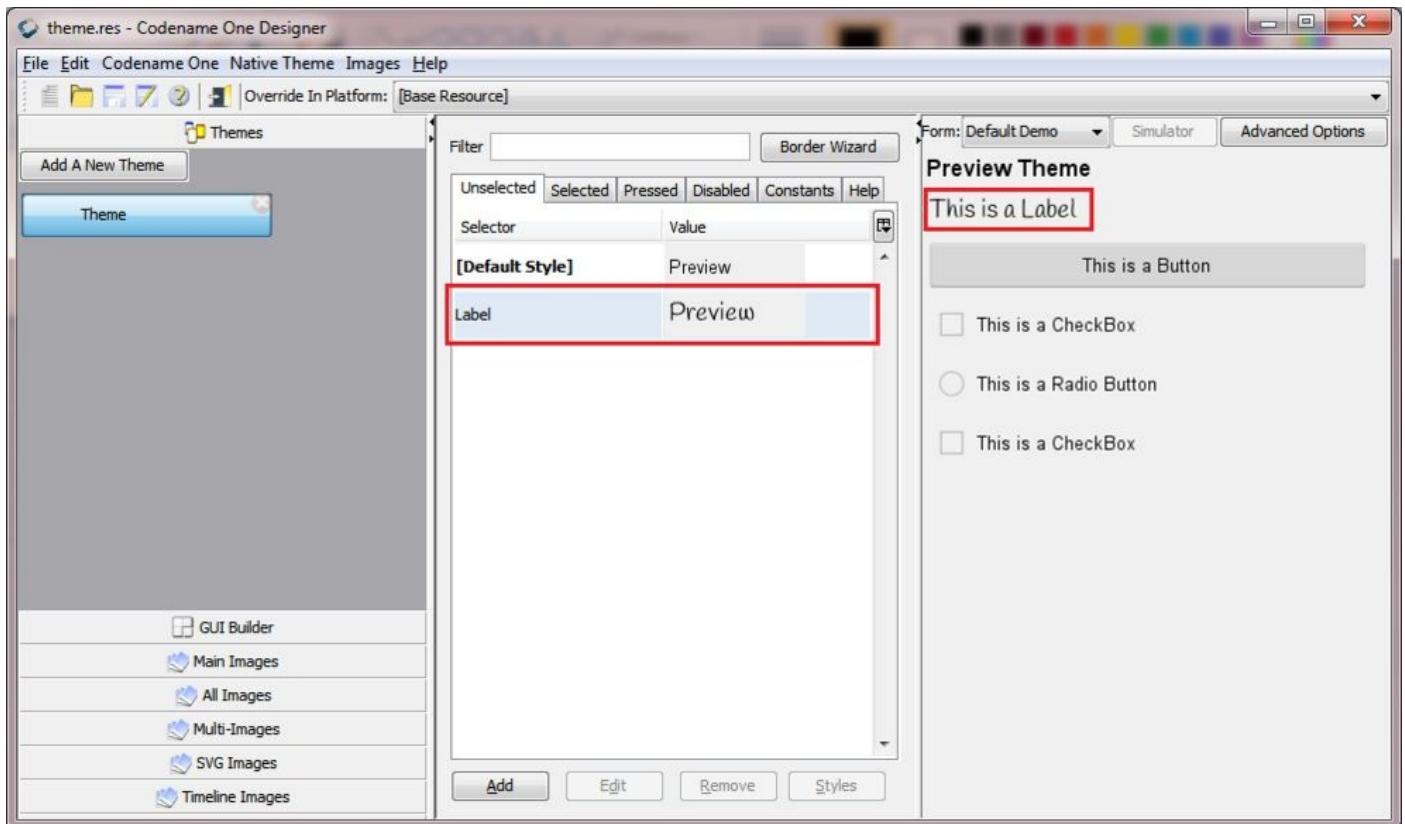


Figure 6.36 : Ajout d'une police True Type Font au composant Label (étape 2)



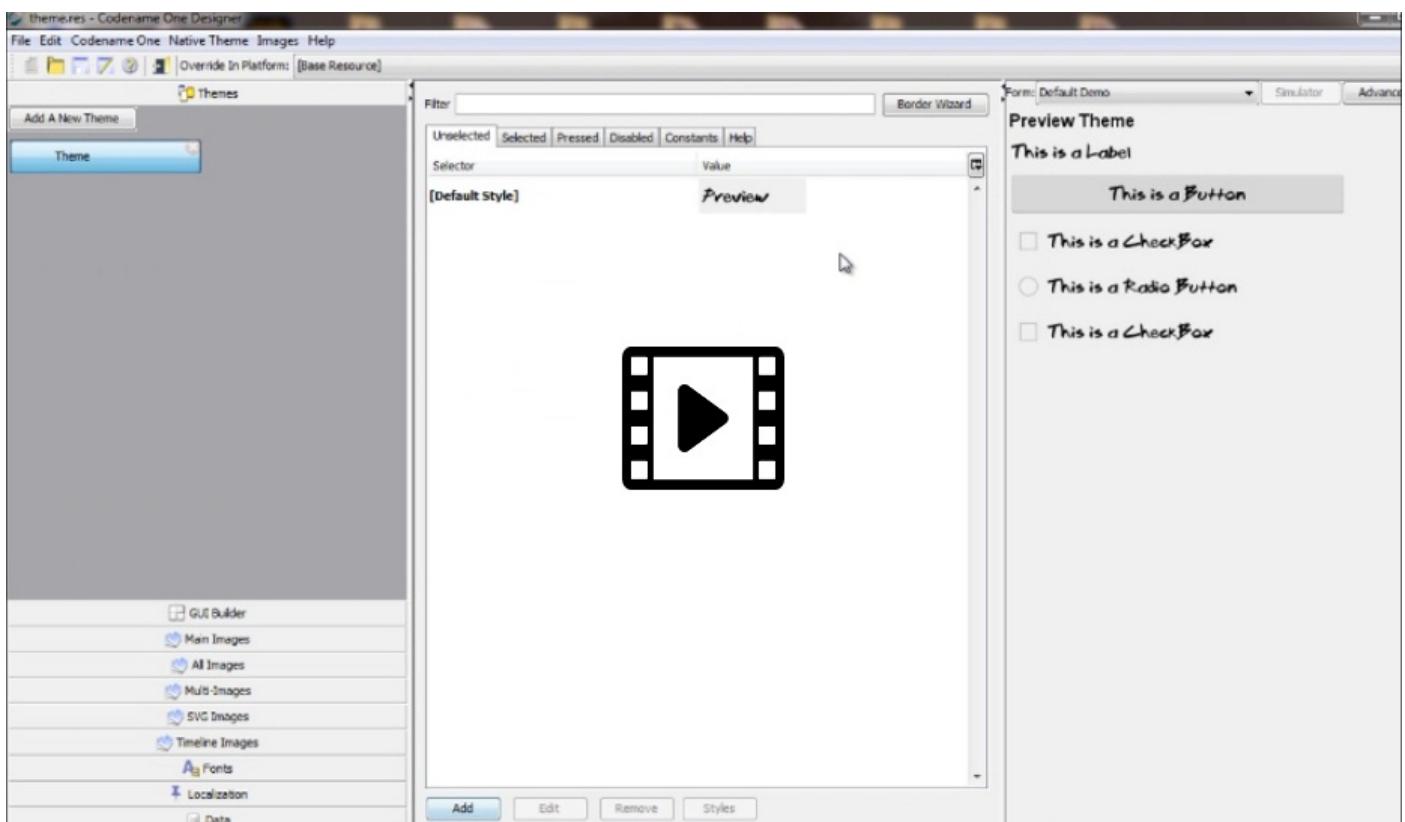
Après validation, l'interface sera semblable à celle de la [Figure 6.37](#).

Figure 6.37 : Ajout d'une police True Type Font au composant Label (étape 3)



Dans le cas que nous venons de voir, la police a été appliquée seulement sur les Label. Pour l'appliquer à tous les composants de l'application qui comportent du texte, retournez sur le thème dans le Designer, double-cliquez sur [Default Style] pour ouvrir la même fenêtre que celle de la [Figure 6.36](#). Répétez, la même opération sous l'onglet Font et validez (voir la vidéo de la [Figure 6.38](#)).

Figure 6.38 : Application d'une police TTF à tous les composants de l'interface (vidéo)



Note > Si vous souhaitez appliquer la police TTF à tous les composants de l'application, répétez l'opération sous les autres onglets (*Selected, Pressed, Disabled*) [et pas seulement sous l'onglet *Unselected*].

7. Internationalisation/localisation

Si vous souhaitez que vos applications soient adoptées par beaucoup de personnes avec un grand nombre de téléchargements alors l'une des choses à faire est de les traduire dans d'autres langues. Dans cette section, nous allons voir comment gérer différentes langues. À titre d'exemple, nous allons concevoir un petit formulaire qui s'affichera en français ou en anglais selon la langue de l'appareil. Au démarrage de l'application, nous commencerons par vérifier la langue par défaut de l'appareil. Si c'est le français, alors le formulaire sera affiché tel quel en français. Dans les autres cas, il sera traduit et affiché en anglais.

Avant de passer à la partie traduction, créons d'abord notre formulaire qui sera constitué d'un champ nom, d'une adresse et d'un numéro de téléphone. Voici le code :

```
public void start() {
    Form f = new Form("Traduction");
    f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

    Label nom=new Label("Nom");
    TextField nomF=new TextField();

    Label adresse=new Label("Adresse");
    TextField adresseF=new TextField();

    Label numero=new Label("Numéro de téléphone");
    TextField numeroF=new TextField();
    numeroF.setConstraint(TextField.NUMERIC);

    Button envoyer=new Button("Envoyer");

    f.addComponent(nom);
    f.addComponent(nomF);
    f.addComponent(adresse);
    f.addComponent(adresseF);
    f.addComponent(numero);
    f.addComponent(numeroF);
    f.addComponent(envoyer);

    f.show();
}
```

Figure 6.39 : Interface à traduire

Traduction

Nom

Adresse

Numéro de téléphone

Envoyer

Maintenant que le formulaire est créé, nous allons mettre en place les textes en anglais correspondant aux Label de l'interface qui sont en français. Pour cela, nous allons créer un bundle de traduction qui va contenir les textes à traduire. Ouvrez le fichier de ressources de votre projet et allez sous l'onglet Localization dans la liste des onglets disponibles sur votre gauche. Cliquez sur le bouton Add Resource Bundle puis dans la zone Name, ajoutez le nom de l'élément (appelé *bundle*) qui va contenir les textes de traduction (ici nommé *MesTraductions* sur la [Figure 6.40](#)). Sélectionnez ce bundle pour afficher son contenu sur la droite de l'interface de traduction (voir [Figure 6.41](#)).

Figure 6.40 : Ajout d'un bundle de traduction

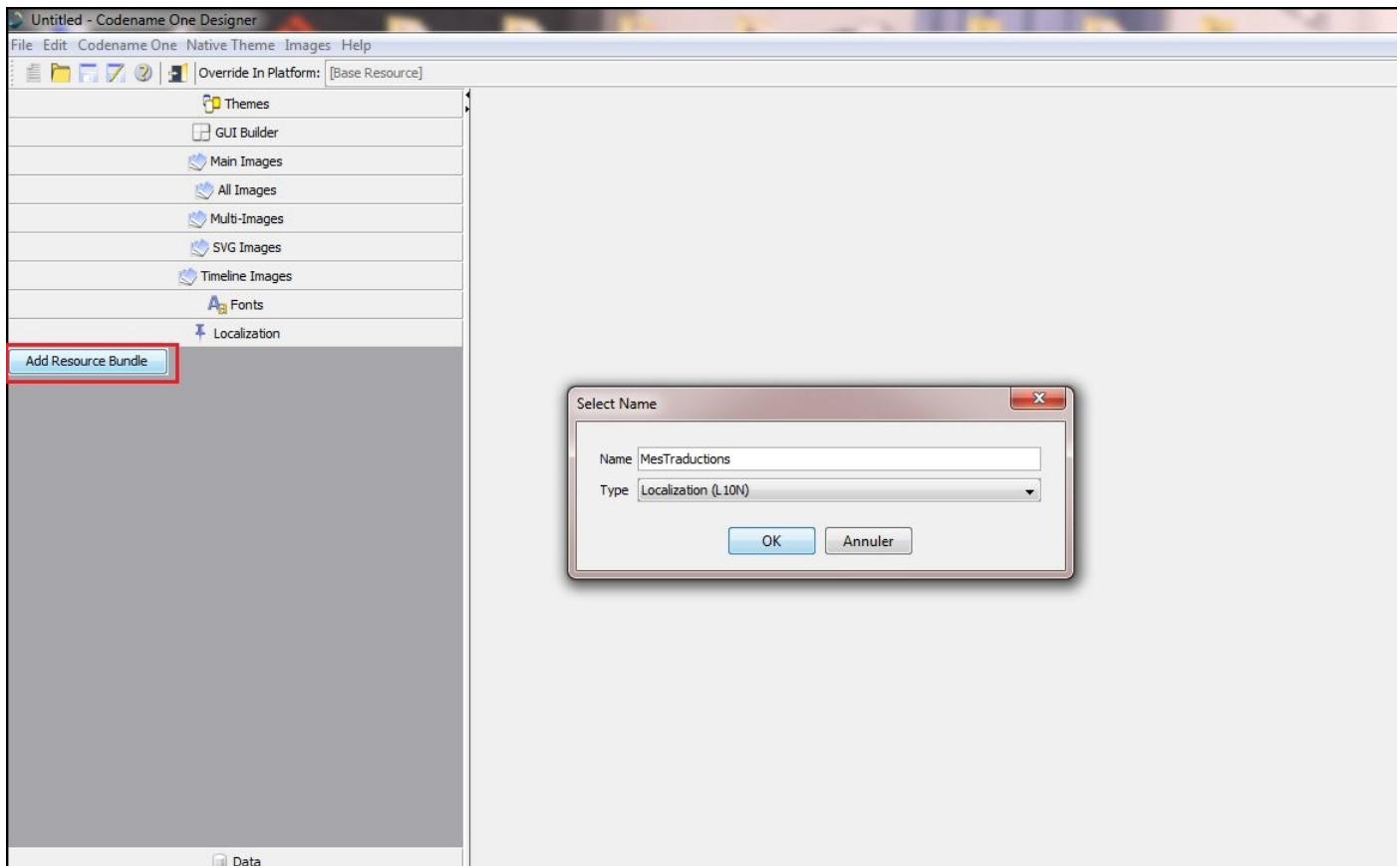


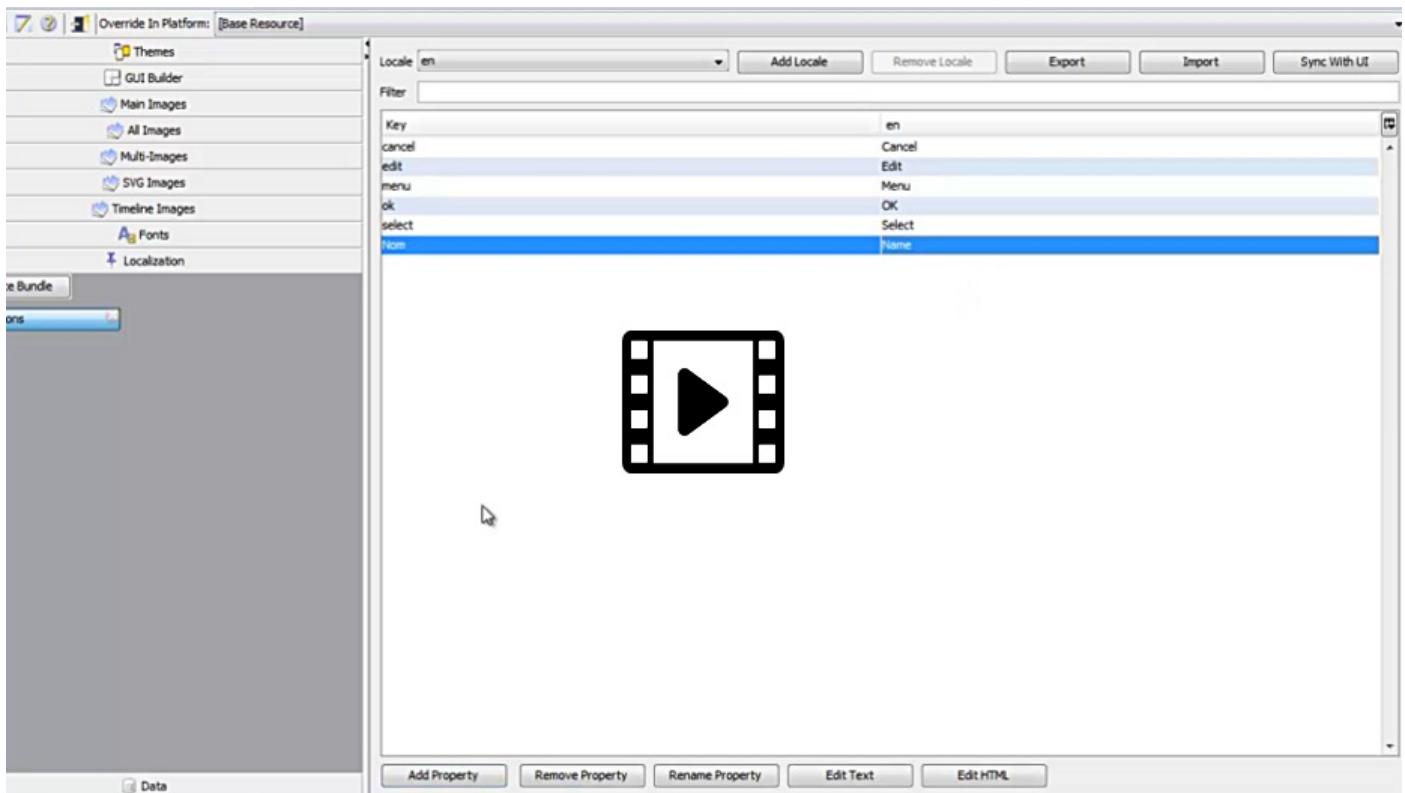
Figure 6.41 : Contenu du bundle créé

This screenshot shows the Localization editor window. At the top, there are buttons for Locale (set to 'en'), Add Locale, Remove Locale, Export, Import, and Sync With UI. Below that is a 'Filter' input field. The main area is a table where French UI keys are mapped to English text. The table has two columns: 'Key' and 'en'. The 'Key' column contains: cancel, edit, menu, ok, and select. The 'en' column contains: Cancel, Edit, Menu, OK, and Select respectively. At the bottom of the editor are buttons for Add Property, Remove Property, Rename Property, Edit Text, and Edit HTML.

Key	en
cancel	Cancel
edit	Edit
menu	Menu
ok	OK
select	Select

Dans la colonne en, nous saisissons les textes anglais en correspondance avec ceux par défaut, en français, de la colonne de gauche (voir la vidéo [ci-dessous](#)).

Figure 6.42 : Ajout de l'équivalent des textes de l'interface en anglais (vidéo)



Vous avez probablement remarqué dans la vidéo que l'on n'a ajouté aucune colonne supplémentaire avant d'effectuer la traduction. C'est parce que la colonne destinée à accueillir les traductions anglaises (celle dont l'en-tête est le mot *en*) est présente par défaut dans tout nouveau bundle. Pour ajouter une nouvelle colonne, qui recueillera donc une nouvelle langue, cliquez sur le bouton Add Locale, donnez-lui un nom puis validez. Une fois créée, vous pouvez y saisir les textes correspondant à cette nouvelle langue. N'oubliez pas d'enregistrer les modifications dans le fichier des ressources. Les [Figure 6.43](#) et [Figure 6.44](#) illustrent l'ajout d'une nouvelle langue (le japonais dont le code ISO 639 est *ja*).

Figure 6.43 : Ajout d'une langue au bundle

The screenshot shows a localization application interface. At the top, there is a toolbar with buttons for 'Locale' (set to 'en'), 'Add Locale' (highlighted with a red box), 'Remove Locale', 'Export', and 'Import'. Below the toolbar is a 'Filter' input field. The main area contains a table with two columns: 'Key' and 'en'. A modal dialog box titled 'Add Locale' is displayed in the center, prompting for a 'Locale Name' (with 'ja' entered) and providing 'OK' and 'Annuler' buttons.

Key	en
Adresse	Address
Envoyer	Send
Nom	Name
Numéro de téléphone	Phone number
cancel	Cancel
edit	Edit
menu	Menu
ok	OK
select	Select

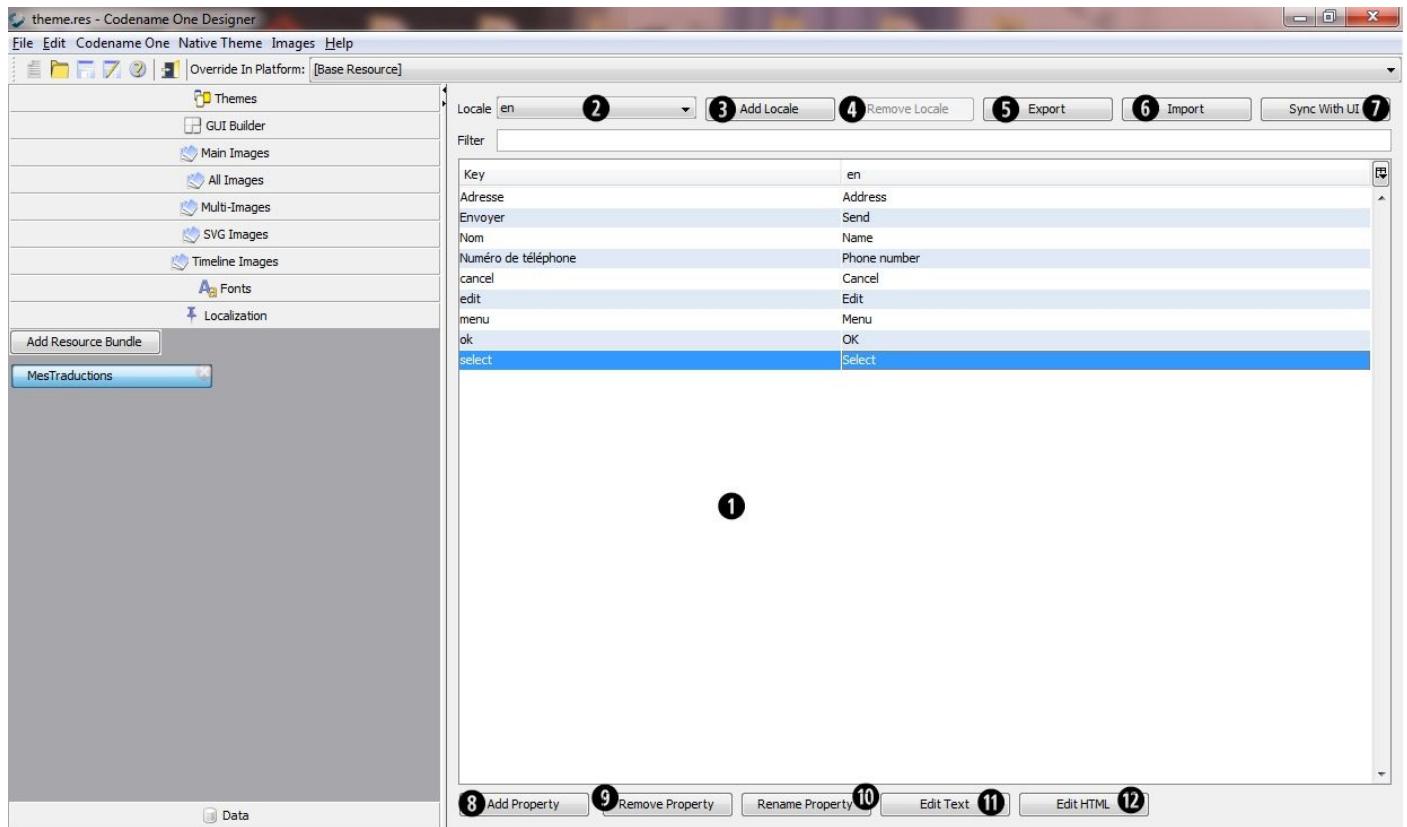
Figure 6.44 : Colonne de la nouvelle langue ajoutée (à compléter)

This screenshot shows the same localization application interface as Figure 6.44, but with a red box highlighting the entire column for the Japanese locale ('ja'). The table structure remains the same, with the 'Key' column and the 'en' column, and the 'ja' column highlighted.

Key	en	ja
Adresse	Address	
Envoyer	Send	
Nom	Name	
Numéro de téléphone	Phone number	
cancel	Cancel	
edit	Edit	
menu	Menu	
ok	OK	
select	Select	

Voici une présentation complète des éléments de l'interface de traduction :

Figure 6.45 : Les éléments de l'interface de traduction



Cette zone contient les clés et les valeurs des chaînes de caractères à traduire. Les clés seront les textes à l'intérieur de votre code Java et les valeurs seront les textes qui les remplaceront. Ces valeurs seront écrites dans les différentes langues dans lesquelles vous voulez traduire votre application.

- ❶ Cette liste représente la liste des langues ajoutées au bundle. Il s'agit d'un mot de deux lettres pour identifier la langue.
- ❷ Permet d'ajouter une nouvelle colonne de langue.
- ❸ Permet de supprimer la langue sélectionnée dans la liste du point ❶.
- ❹ Permet d'exporter les clés et les valeurs sous forme de fichier CSV, XML ou dans un format propriétaire de Codename One.
- ❺ Permet d'importer les clés et les valeurs d'un fichier CSV, XML ou du format propriétaire de Codename One vers le Designer.
- ❻ Permet de se synchroniser avec les pages créées avec le GUI builder pour y extraire et afficher toutes les chaînes de caractères sous forme de clés pour la traduction.
- ❼ Permet d'ajouter une nouvelle clé.
- ➍ Permet de supprimer une clé.

⑩ Permet de renommer une clé.

⑪ Permet d'éditer la valeur d'une clé.

⑫ Permet d'éditer la valeur d'une clé avec des attributs HTML.

Voyons maintenant comment écrire le code qui nous permettra de détecter la langue du téléphone et de charger les données anglaises si cette langue est différente du français. Placez ce code dans la méthode `init(Object)` comme ceci :

```
//...
private Resources theme;
//...
public void init(Object context) {
    try {
        theme=Resources.openLayered("/theme");
        UIManager.getInstance().setThemeProps(
            theme.getTheme(theme.getThemeResourceNames()[0]));
    } catch(IOException e){
        e.printStackTrace();
    }
}

L10NManager lm=Display.getInstance().getLocalizationManager();
```

① String langue=lm.getLanguage(); ② if(!langue.equals("fr")) { ③ Hashtable loc=theme.getL10N("MesTraductions", "en"); ④ UIManager.getInstance().setBundle(loc); ⑤ } }

① Création d'un objet L10NManager. Cette classe contient tout ce qu'il faut pour gérer la localisation et l'internationalisation.

② Récupération de la langue locale de l'appareil avec la méthode `getLanguage()` de la classe L10NManager. La valeur de cette langue sera constituée uniquement de deux lettres de la norme ISO 639. Ex : `fr` pour le français, `en` pour l'anglais, etc.

③ Ici, nous effectuons une vérification pour voir si la langue est différente du français.

À ce niveau, nous faisons appel à la méthode `getL10N()` de l'objet `theme` (de la classe Resource qui nous donne accès aux contenus du fichier de ressources) pour accéder ④ aux contenus du bundle `MesTraductions`. En premier paramètre à `getL10N()`, le nom du bundle à utiliser et en deuxième paramètre la langue à charger. Cette méthode retourne un Hashtable contenant sous forme de clés et de valeurs des textes traduits.

Appel de la méthode `setBundle()` de UIManager pour effectuer à la volée la ⑤ traduction des textes de l'interface ajoutés au bundle `MesTraductions`. Elle prend en paramètre le Hashtable retourné par `getL10N()`.

La [Figure 6.46](#) montre l'interface de notre formulaire traduit en anglais tel qu'il s'affiche au lancement de l'application une fois la langue de l'appareil identifiée comme étant autre

que le français.

Figure 6.46 : Interface traduite (en anglais)

The image shows a user interface for a translation application. At the top, the word "Traduction" is displayed in bold black text. Below this, there are three input fields with labels: "Name", "Address", and "phone number", each followed by an empty text input box. At the bottom center of the screen, there is a blue "Send" button.

8. Fonctionnalités diverses

8.1. Protection par mot de passe

Il est possible de protéger un fichier de ressources par un mot de passe. En revanche, attention, il n'existe pas de moyen de retrouver ce mot de passe si vous l'oubliez. Je vous conseille donc de conserver une copie non protégée du fichier. Pour ajouter cette protection, allez dans le menu File du Designer et cliquez sur Set Password (voir [Figure 6.47](#)). Entrez ensuite le mot de passe dans le champ Password puis de nouveau dans le champ Confirm (voir [Figure 6.48](#)) et validez.

Astuce > Pour enlever le mot de passe d'un fichier de ressources, définissez comme mot de passe un champ vide puis validez.

Figure 6.47 : Menu d'ajout de mot de passe au fichier de ressources

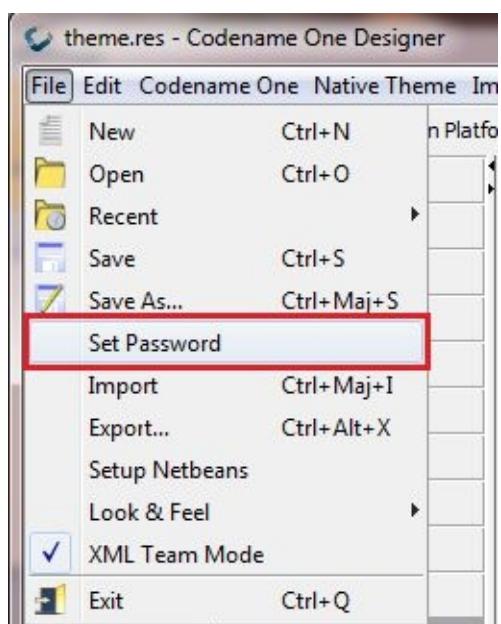


Figure 6.48 : Interface d'ajout de mot de passe au fichier de ressources



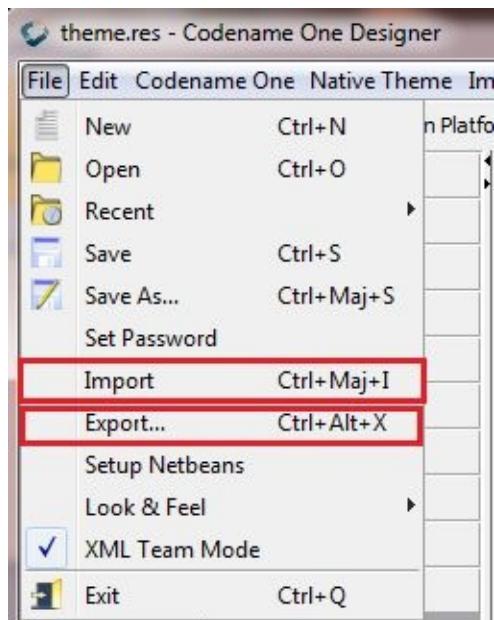
8.2. Exporter et importer depuis un autre fichier de ressources

Il est possible d'extraire dans un dossier tous les fichiers (images, texte de traduction, etc...) contenus dans un fichier de ressources. Mieux encore, vous pouvez importer des données (thèmes, images et autres) d'un fichier de ressources vers un autre.

Pour l'exportation, allez dans le menu File, cliquez sur Export, choisissez un dossier où les fichiers seront extraits puis validez.

Pour l'importation, cliquez sur Import toujours dans le menu File puis choisissez le fichier .RES duquel vous voulez importer les données.

Figure 6.49 : Menus d'importation et d'exportation



Plug-ins et code natif

Codename One étant une API qui supporte diverses plateformes, elle ne peut nécessairement pas intégrer toutes les fonctionnalités de chacune. Pour des raisons de popularité, certaines plateformes ont la priorité (iOS et Android par exemple) sur d'autres, mais ce n'est pas irrémédiable. Que vous souhaitiez accéder à des services en ligne ou simplement ajouter des fonctionnalités natives non intégrées à Codename One, vous pouvez recourir à des [plug-ins](#) et intégrer du [code natif](#). C'est ce que nous allons voir dans ce chapitre.

1. Le système de plug-in de Codename One (CN1LIB)

Codename One permet de packager sous forme de plug-in des fonctionnalités qui ne sont pas disponibles par défaut dans son API. Ce plug-in est un fichier d'extension .cn1lib, qui peut être ajouté simplement à n'importe quel projet. En fait, ce n'est rien d'autre qu'un fichier compressé contenant du code compilé. Vous pouvez l'ouvrir avec n'importe quel logiciel de compression et de décompression (WinRAR par exemple). Il peut soit contenir uniquement du code Java compilé soit être mélangé à du code natif propre aux SDK des plateformes supportées.

Créer des fonctionnalités sous forme de fichier .cn1lib permet aussi de rendre un projet modulable. Vous pourrez ainsi réutiliser le code de ce fichier dans d'autres projets de votre choix.

1.1. Des plug-ins gratuits à votre disposition

Des développeurs de la communauté de Codename One ont créé des plug-ins pour simplifier l'accès à certaines fonctionnalités et à certains services courants en ligne. Ceux-ci sont disponibles sur le site officiel à la page [CN1Lib's](#). Des ajouts sont souvent effectués, donc pensez à y faire régulièrement un tour pour bénéficier des nouveautés. Voici un récapitulatif de quelques-uns jugés utiles et plus aboutis.

Note > Les plug-ins présentés ici ne sont pas fonctionnels sur toutes les plateformes supportées par Codename One. Certains ne fonctionnent que sous iOS et Android.

Tableau 7.1 : Liste de plug-ins créés par la communauté de Codename One

Nom	Description
CN1Sockets	Permet de manipuler des sockets TCP.
Dropbox	Donne accès aux fonctionnalités du service de stockage de données en ligne Dropbox.
PubNub	Permet de faire communiquer des applications entre elles en utilisant le service de communication PubNub.
Google Maps	Permet d'intégrer Google Maps à une application.
Toast	Permet d'afficher des notifications nommées Toast sous Android.
ComScore	Donne accès à l'API de données analytiques de ComScore.
Calendar	Donne accès au calendrier natif de l'appareil.
CN1OfflineMaps	Permet d'utiliser le composant MapComponent sans connexion internet.

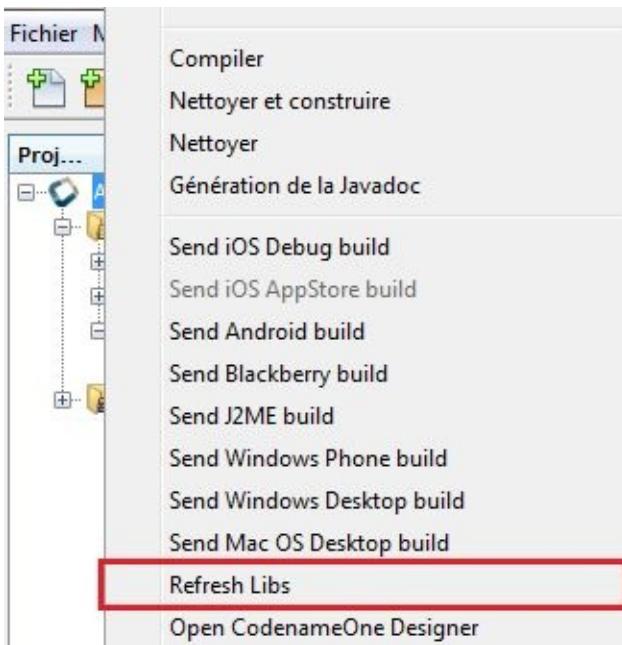
CN1Regex	Donne accès à l'utilisation des expressions régulières.
CN1Charts	Permet de créer des diagrammes. Plug-in basé sur une API écrite en HTML5.
CN1aChartEngine	Plug-in basé sur la bibliothèque aChartEngine. Permet de créer des diagrammes sans l'utilisation d'un navigateur web comme c'est le cas avec CN1Charts. L'API de création des diagrammes intégrée en interne à Codename One est basée sur cette bibliothèque.
Bouncy castle crypto	Donne accès aux fonctionnalités de la célèbre bibliothèque de cryptographie Bouncy Castle.
CN1JTar	Donne accès aux fonctionnalités de JTar (bibliothèque de compression et de décompression de données).
CN1JSON	Permet de manipuler des fichiers JSON.
Flurry Ads et Flurry Analytics	Permet l'intégration de la publicité plein écran en plus de donner accès à des données statistiques fournies par Flurry.
Sensors	Permet d'utiliser des fonctionnalités d'accéléromètre.
Core2D	Fournit des fonctionnalités pour la conception de jeux en 2D. Gestion de sprites, de tiles, etc.

Ces plug-ins sont livrés avec leur code source, donc vous pouvez les modifier ou contribuer au plug-in si l'aventure vous tente. Faites-le si vous le pouvez, la communauté ne pourra que vous en être reconnaissante.

1.2. Utilisation d'un plug-in

Voyons maintenant comment utiliser un plug-in Codename One quelconque téléchargé sur le Web. C'est très simple. Copiez le fichier du plug-in (le fichier .cn1lib téléchargé) dans le dossier lib de votre projet, cliquez droit et sélectionnez l'entrée Refresh Libs (voir la [Figure 7.1](#)). Et voilà ! Vous êtes maintenant prêt à utiliser le plug-in. Il vous suffit de lire l'aide fournie sur la page qui l'héberge et vous pourrez utiliser des fonctionnalités qu'il met à votre disposition.

Figure 7.1 : Intégrer un plug-in



Pour bien illustrer cela, nous allons écrire un exemple avec un plug-in qui permet de créer un toast. Un *toast* est une fonctionnalité qui permet de notifier un utilisateur sans le déranger même s'il utilise une autre application que la vôtre. C'est un message non modal qui ne s'affiche que durant quelques secondes. Pour ce test, téléchargez le projet NetBeans du plug-in sur GitHub [Pmovil/Toast](#). Compilez-le et récupérez le fichier .cn1lib dans le dossier dist du projet. Vous trouverez aussi le fichier du plug-in dans les sources du livre.

Créez un nouveau projet (nommé TestPlugin par exemple) et déposez dans le dossier lib de ce projet le fichier .cn1lib récupéré après compilation. Faites ensuite un clic droit sur le projet puis sélectionnez Refresh Libs comme vu précédemment. Voici le code de notre exemple.

Note > Ce plug-in fonctionnera sous Android, iOS et BlackBerry.

Exemple 7.1 : Exemple d'utilisation d'un plug-in

```
public class TestPlugin {

    private Form current;
    private Object mContext;

    public void init(Object context) {
        mContext=context;

        try {
            Resources theme = Resources.openLayered("/theme");
            UIManager.getInstance().setThemeProps(
                theme.getTheme(theme.getThemeResourceNames()[0]));
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    public void start() {
        if(current != null){
```

```

        current.show();
    return;
}

Form f=new Form("Test Plugin");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

Button b=new Button("Affiche un Toast");
b.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        Toast.makeText(mContext, "Coucou, je suis là ;-)",
Toast.LENGTH_LONG).show();
    }
}); f.addComponent(b); f.show(); } public void stop() { current =
Display.getInstance().getCurrent(); } public void destroy() { } }
```

Appel de la méthode `makeText()` de la classe `Toast`. Cet appel est suivi de celui de la méthode `show()` qui permet de lancer l'affichage. `makeText()` prend en premier paramètre l'objet en paramètre de la méthode `init()` ; en deuxième paramètre, le texte à afficher ; et en dernier paramètre, la durée d'affichage du texte. Ce dernier peut prendre l'une des valeurs suivantes : `Toast.LENGTH_LONG` (pour un affichage long), `Toast.LENGTH_SHORT` (pour un affichage court). Comme vous le savez déjà, la classe `Toast` que nous utilisons ici se trouve dans le plug-in que nous avons ajouté au projet.

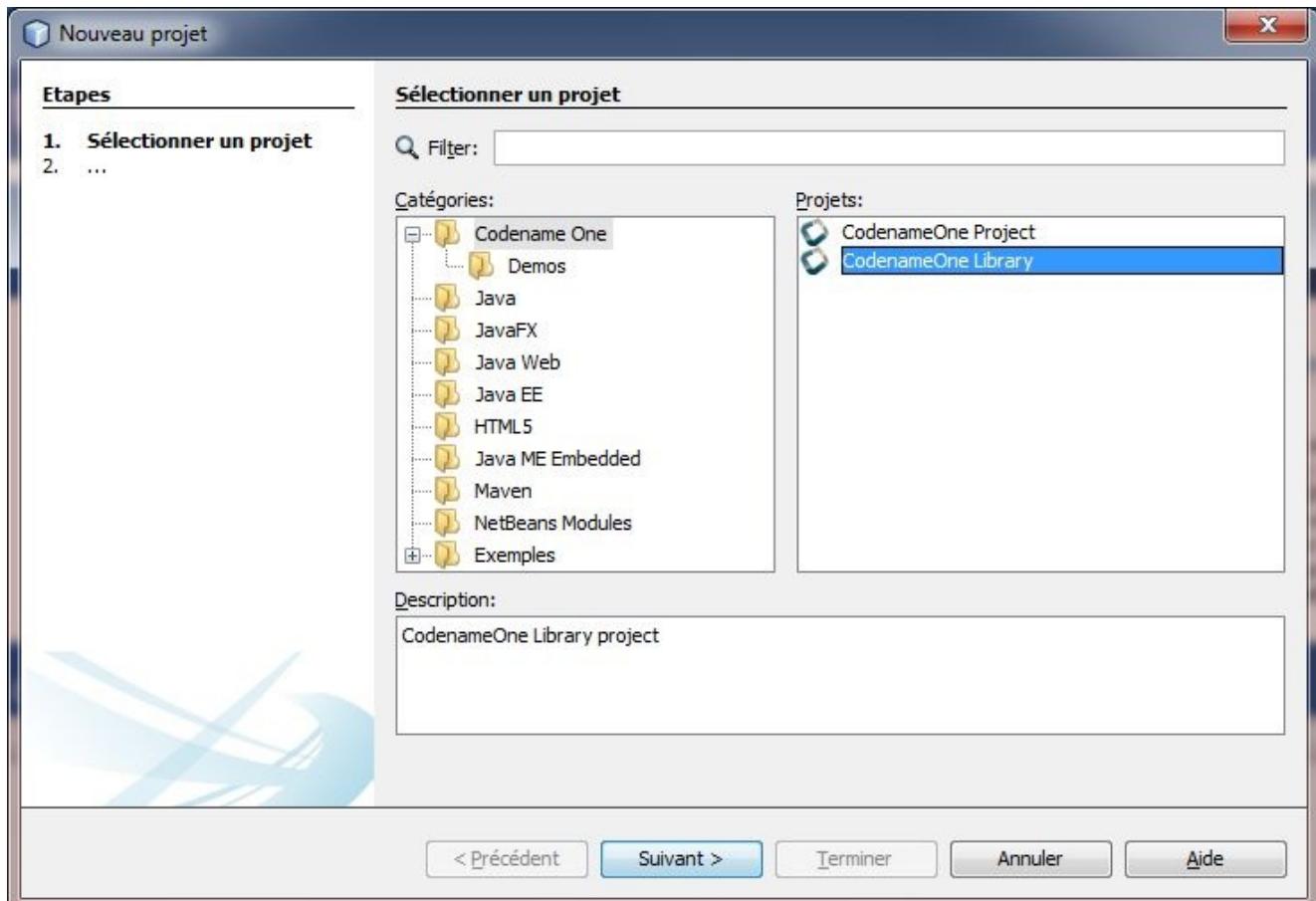
Figure 7.2 : Un test de toast sous Android



1.3. Crédit d'un plug-in

Créer un plug-in n'est pas plus compliqué qu'en utiliser un si vous savez ce que vous voulez faire. Son code peut être constitué uniquement de code Java ou être un mélange de code Java et de code natif (voir [Section 2, Code natif \(au-delà des limitations\)](#)). Suivez la procédure suivante pour créer votre propre plug-in.

1. Sous NetBeans, cliquez sur Nouveau Projet. Sous la catégorie Codename One, sélectionnez CodenameOne Library et terminez la création du projet après avoir donné un nom au plug-in.



2. Après validation, le projet est créé et une classe dont le contenu est semblable à celui de la figure suivante est générée. Contrairement à la création d'un projet Codename One classique, nous n'avons pas eu à entrer le nom du package ni le nom de la classe qui ont été créés par défaut. Vous pouvez modifier ces informations dans le code généré si vous voulez.

```

1  /*
2   * Copyright comment here
3   */
4  package com.codename1.hello;
5
6  /**
7   * This is a demo class to help you get started building a library
8   *
9   * @author Your name here
10  */
11 public class FirstCodenameOneLibrary {
12     /**
13      * Method javadoc information
14      */
15     public void hello() {
16         System.out.println("Hello");
17     }
18 }
19

```

3. Codez maintenant comme d'habitude en vous servant de la classe générée ou en ajoutant d'autres classes à votre projet si nécessaire. Une fois terminé et le projet compilé, rendez-vous dans le dossier dist puis récupérez votre fichier .cn1lib.

2. Code natif (au-delà des limitations)

Même si le but de Codename One en tant que framework multiplateforme est de supporter un maximum de fonctionnalités sur le plus de plateformes possibles, il est normal que certaines manquent dans l'API. Pour résoudre ce problème, un pont a été créé pour faire communiquer du code Java écrit en Codename One avec du code natif des SDK des différentes plateformes supportées. Ainsi, vous pouvez écrire des fonctionnalités en Objective C ou avec l'API Java d'Android, de BlackBerry, etc., et les appeler depuis le code Java écrit en Codename One.

Attention > Pour tirer profit de ce que nous allons voir dans cette section, vous devez savoir utiliser au minimum l'API natif du SDK de l'une des plateformes supportées par Codename One. Vous en aurez besoin pour implémenter dans vos applications les fonctionnalités manquantes dans l'API interne.

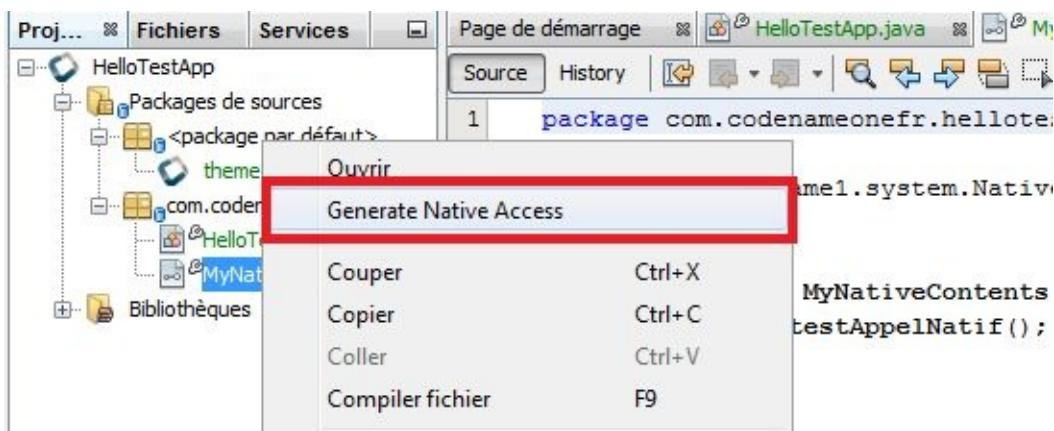
Vous pouvez utiliser ce pont dans un projet d'application ou de création de plug-ins. Pour tester cela, créez un projet constitué d'une fenêtre et d'un bouton. Un clic sur le bouton fera appel à une méthode qui exécutera le code natif de votre choix. Une fois le projet créé, ajoutez une classe d'interface au projet et nommez-la comme vous voulez (MyNativeContents.java par exemple). Cette interface doit obligatoirement hériter de la classe NativeInterface. Ajoutez-lui le prototype d'une méthode qui effectuera l'action que vous voulez exécuter (voir la [Figure 7.3](#)).

Figure 7.3 : Appel de code natif

```
3 package com.codenameonefr.hellotestapp;
4
5 import com.codename1.system.NativeInterface;
6
7 |
8 public interface MyNativeContents extends NativeInterface {
9     public void testAppelNatif();
10 }
11
```

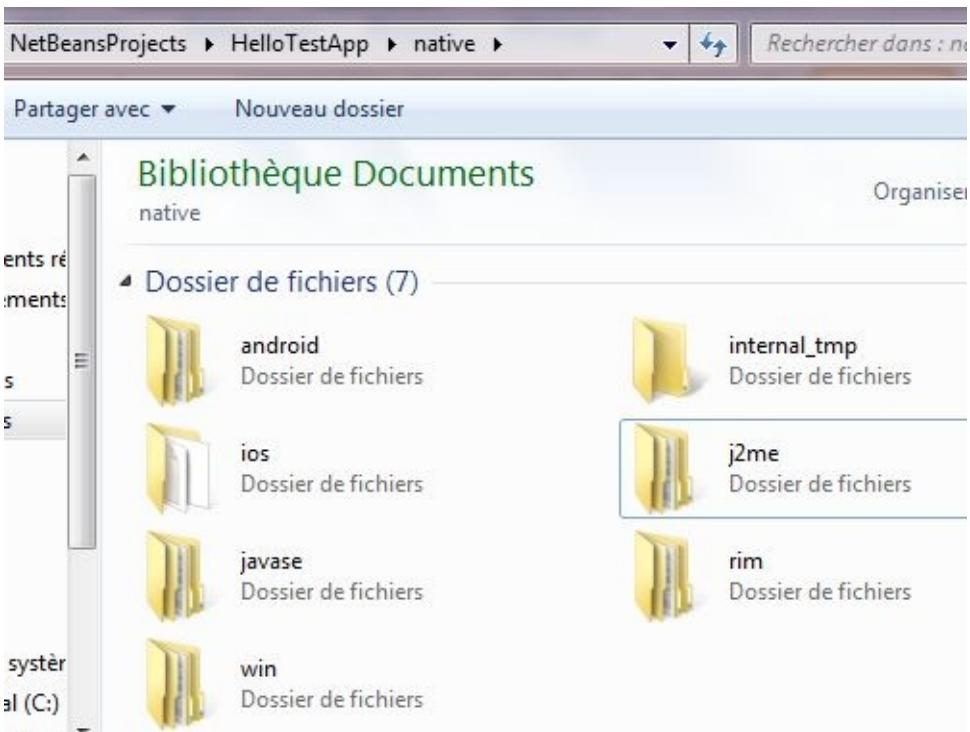
Dans l'explorateur du projet, cliquez droit sur le fichier de l'interface (MyNativeContents.java dans notre cas) et sélectionnez l'entrée Generate Native Access (voir la [Figure 7.4](#)).

Figure 7.4 : Génération de classes natives



Cela va générer automatiquement des classes (pour Android, BlackBerry, Windows phone et J2ME) et un fichier d'extension .m et .h (pour iOS). Ces fichiers sont stockés dans des sous-dossiers du dossier native de votre projet. Ils portent le nom des différentes plateformes mobiles (voir [Figure 7.5](#)).

Figure 7.5 : Sous-dossiers contenant du code natif



Les fichiers générés automatiquement dans ces dossiers sont ceux dans lesquels vous allez écrire vos codes (avec l'éditeur de code adapté au langage ou dans NetBeans dans notre cas) en utilisant l'API natif de la plateforme qui vous intéresse. L'API des plateformes Android, BlackBerry et J2ME étant en Java, la structure du code généré est la même pour ces trois plateformes. Celle de Windows Phone est en C# et celle de l'iOS est en Objective C.

Exemple 7.2 : Code du fichier d'implémentation pour Android, BlackBerry, J2ME

Note > Il s'agit du fichier `MyNativeContentsImpl.java` se trouvant dans les dossiers `android`, `rim` et `j2me` du dossier `native`.

```

package com.codenameonefr.hellotestapp;

public class MyNativeContentsImpl {
    public void testAppelNatif() {
        //Écrire ici le code natif Java du SDK d'Android de BlackBerry OS
        ou de J2ME de ce que vous voulez faire
    }

    public boolean isSupported() {
        return false;
    }
}

```

Exemple 7.3 : Code du fichier d'implémentation pour Windows Phone

Note > Il s'agit du fichier *MyNativeContentsImpl.cs*) se trouvant dans le dossier *win* du dossier *native*.

```

namespace com.codenameonefr.hellotestapp {

using System;
using System.Windows;

public class MyNativeContentsImpl {
    public void testAppelNatif() {
        //Écrire ici le code natif C# de ce que vous voulez faire
    }

    public bool isSupported() {
        return false;
    }
}
}

```

Exemple 7.4 : Code du fichier d'implémentation pour iOS

Note > Il s'agit des fichiers

com_codenameonefr_hellotestapp_MyNativeContentsImpl.m et
com_codenameonefr_hellotestapp_MyNativeContentsImpl.h) se trouvant dans le dossier *ios* du dossier *native*.

```

#import "com_codenameonefr_hellotestapp_MyNativeContentsImpl.h"
@implementation com_codenameonefr_hellotestapp_MyNativeContentsImpl
-(void)testAppelNatif{
    //Écrire ici le code natif Objective C de ce que vous voulez faire
}
-(BOOL)isSupported{
    return NO;
}
@end

```

```
#import <Foundation/Foundation.h>
@interface com_codenameonefr_helloworldapp_MyNativeContentsImpl : NSObject {
}
-(void)testAppelNatif;
-(BOOL)isSupported;
@end
```

Une fois que vous aurez implémenté la méthode `testAppelNatif()` dans un ou plusieurs de ces fichiers, revenez au code source Java du fichier principal pour l'appeler. Voici le code de l'action de clic sur le bouton de l'application de test.

```
public void start() {

    Form f=new Form("Native Interface Test");
    f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

    Button b=new Button("Clique ici");
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {

            MyNativeContents nc=
            (MyNativeContents)NativeLookup.create(MyNativeContents.class);

❶ nc.testAppelNatif(); ❷ } });
    f.addComponent(b);
    f.show();
}
```

- ❶ Création d'une instance de l'interface `MyNativeContents`. Cette création se fait avec la méthode statique `create()` de la classe `NativeLookup`.
- ❷ Appel de la méthode `testAppelNatif()`.

C'est enfin prêt. Après compilation, vous pouvez maintenant tester l'appel de votre code natif en touchant le bouton placé sur la fenêtre de l'application.

Attention > *Le test d'une application Codename One utilisant le NativeInterface doit être effectué directement sur le téléphone et non sur le simulateur qui ne pourra pas l'exécuter.*

Monétisation

Au delà du désir d'apporter une solution à un problème ou de distraire avec un jeu, une application mobile peut aussi être créée à des fins lucratives. À vous de décider de la manière dont vous voulez la distribuer. Vous pouvez la vendre, l'offrir gratuitement sans rien gagner ou encore l'offrir gratuitement, mais gagner quand même de l'argent en y insérant de la publicité ou en vendant des fonctionnalités supplémentaires à l'intérieur de l'application. Si vous avez opté pour ce troisième choix, alors ce chapitre vous aidera à le mener à bien.

1. Monétisation des applications gratuites

Codename One supporte divers réseaux publicitaires assez connus pour vous aider à monétiser vos applications gratuites. Ces réseaux permettent d'insérer des publicités à l'intérieur des applications. Chaque fois qu'elles s'affichent ou que l'utilisateur clique dessus, cela génère des revenus qui peuvent devenir conséquents si l'application est téléchargée un grand nombre de fois.

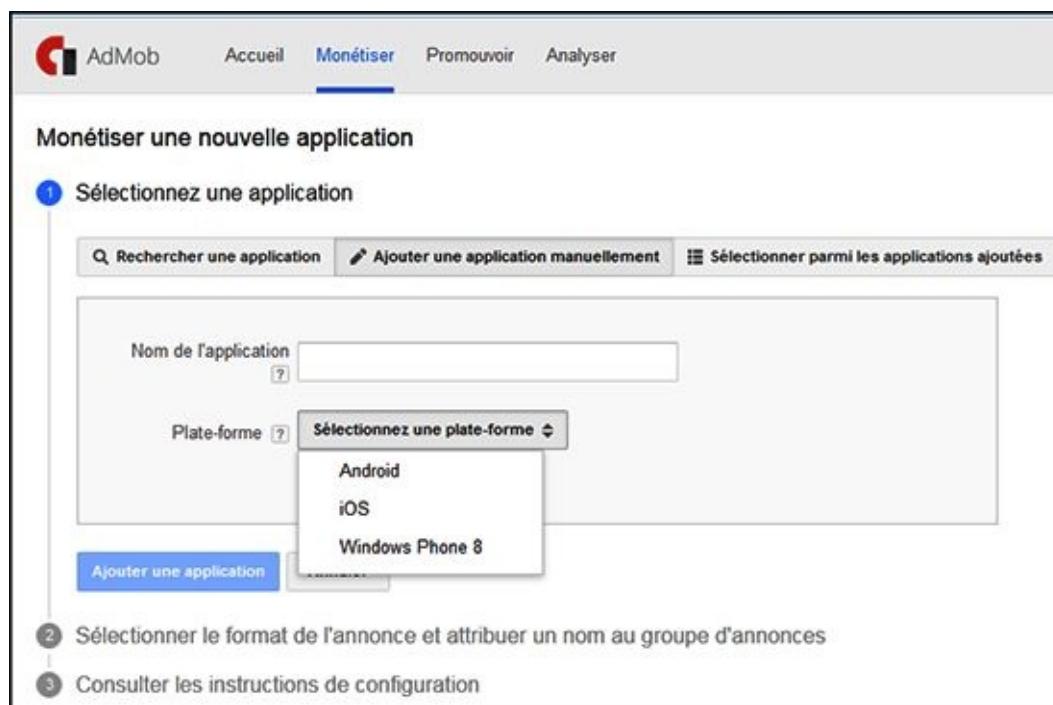
1.1. Google Mobile Ads

Google Mobile Ads est un service de monétisation créé par Google et disponible à travers Google AdMob (l'un des principaux réseaux publicitaires à destination des plateformes mobiles). Il permet de monétiser les applications mobiles à l'aide de bannières ou de publicités s'affichant en plein écran. Seule la publicité basée sur les bannières est disponible dans l'API interne de Codename One. Pour déployer des publicités en plein écran, vous devez recourir à un plug-in spécifique (Voir l'encadré [plus loin](#)).

Contrairement à ce qu'on pourrait croire, Google Mobile Ads n'est pas seulement supporté par les applications Android, mais aussi par les applications iOS et Windows Phone. Pour utiliser ce service, vous devez vous inscrire sur le site d'[AdMob](#). Une fois l'inscription effectuée, identifiez-vous puis rendez-vous sous l'onglet Monétiser (voir [Figure 8.1](#)).

Note > *Même si cette régie publicitaire est aussi disponible sous Windows Phone, Codename One ne la supporte que sur Android et iOS.*

Figure 8.1 : Enregistrement d'une application sur AdMob



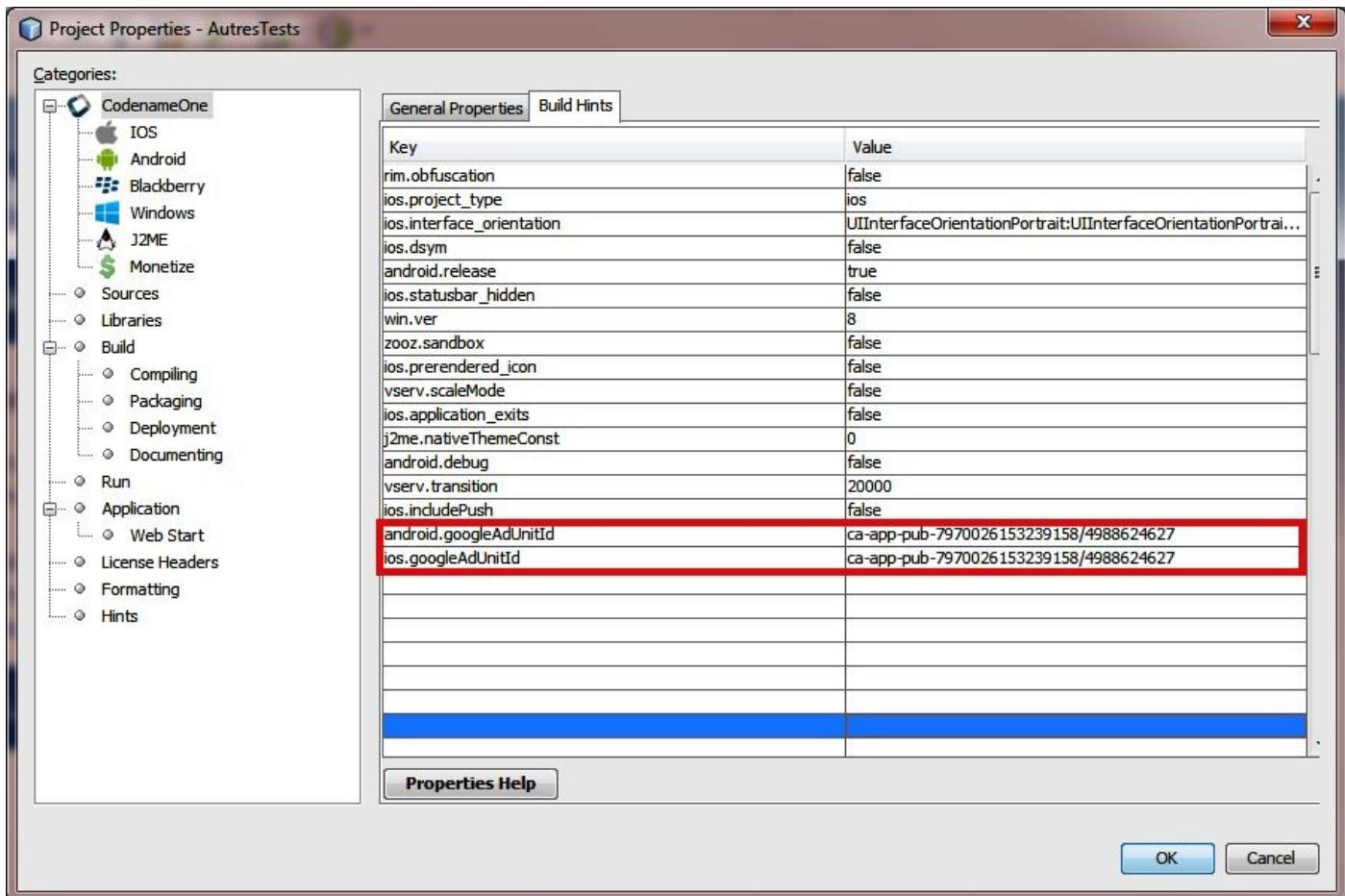
Remplissez les formulaires des trois étapes proposées et récupérez (en copiant) l'identifiant ID du bloc d'annonces (voir [Figure 8.2](#)) créé. Nous allons devoir l'enregistrer dans l'application.

Figure 8.2 : Récupération de l'identifiant du bloc publicitaire



Dans NetBeans, ouvrez la fenêtre de propriétés du projet de votre application (clic droit sur son nom puis entrée Propriétés) et sélectionnez l'onglet Build Hints. Vous devez ajouter à la liste le nom de la clé correspondant à la plateforme visée avec pour valeur l'identifiant copié précédemment. Pour Android, elle s'appellera `android.googleAdUnitId` ; et pour iOS, `ios.googleAdUnitId`. Si vous visez les deux plateformes, ajoutez les deux clés (voir l'encadré sur la [Figure 8.3](#)). Dans ce dernier cas, la publicité correspondant à chacune des deux plateformes sera générée et s'affichera en dessous des fenêtres de votre application. Validez et compilez, puis consultez régulièrement votre tableau de bord pour surveiller vos revenus.

Figure 8.3 : Enregistrement dans l'application de la clé du bloc publicitaire



Encadré : Publicité en plein écran (Admob Interstitial Ads)

Le support de la publicité plein écran de Google Mobile Ads n'est pas intégré à l'API interne, mais vous pouvez l'ajouter à l'aide du plug-in [Admob Interstitial](#). Téléchargez son fichier .cn1lib, ajoutez-le à votre projet comme nous l'avons vu au chapitre [Plugins et code natif](#), [Section 1.2, Utilisation d'un plug-in](#), puis passez à l'écriture du code. Ce dernier peut être placé à l'intérieur de la méthode `init(Object)` ou au début de la méthode `start()`. Voici un exemple :

```
//...
public void init(Object context) {
//...
AdMobManager admobManager=new AdMobManager("ID pour la plateforme
Android ou iOS");
```

❶ `admobManager.loadAd();` ❷ `admobManager.showAdIfLoaded();` ❸ } //...

Création de l'objet `AdMobManager`. En constructeur à cette classe, nous ajoutons la ❶ clé de la plateforme à viser. Cette clé est celle que vous aviez récupérée dans votre tableau de bord de Google Admob.

❷ Appel de la méthode `loadAd()` pour charger une publicité.

❸ Appel de la méthode `showAdIfLoaded()` qui affiche la publicité si elle est chargée.

Les actions de chargement et d'affichage de la publicité peuvent aussi s'effectuer en utilisant une seule méthode qui combinent les deux actions. Il s'agit de la méthode `loadAndShow()`.

Pour finir l'intégration, nous devons retourner dans les propriétés du projet et y ajouter quelques arguments comme nous l'avons fait pour les bannières. Sous l'onglet Build Hints, ajoutez les arguments et valeurs suivants :

pour Android :

- `android.xapplication =`
`<meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />`
`<activity android:name="com.google.android.gms.ads.AdActivity" android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|uiMode|screenSize|smallestScreenSize" />`
- `android.includeGPlayServices = true`

pour iOS :

- `ios.add_libs = AdSupport.framework;SystemConfiguration.framework;CoreTelephony.framework`
- `ios.objC = true`

Figure 8.4 : Aperçu d'une bannière AdMob



1.2. Inneractive

Tout comme Google Mobile Ads, Inneractive est également une régie publicitaire assez

connue. Elle fournit des bannières que vous pouvez intégrer à vos applications. Pour cela, vous devez d'abord vous inscrire sur [leur site](#) (en anglais), créer une publicité et récupérer son ID. Vous devez ensuite initialiser Inneractive en plaçant le code suivant dans la méthode `init(Object)` ou `start()` de votre projet.

```
AdsService.setAdsProvider(InnerActive.class);
```

Une fois ce code inséré, vous pouvez placer la bannière de la publicité créée à l'endroit que vous voulez sur un Form quelconque de votre application. Le composant à utiliser pour l'affichage de cette bannière est le composant Ads. Il contient une méthode nommée `setAppID(String appID)` qui prend en paramètre l'ID que vous avez récupéré dans votre tableau de bord sur le site d'Inneractive. Vous pouvez aussi passer cet ID en paramètre au constructeur de la classe Ads. Une fois l'ID ajouté avec cette méthode, vous pouvez ajouter l'objet Ads créé sur un composant Form ou faire d'autres paramétrages avec d'autres méthodes de la classe Ads avant l'ajout. Le code de la création et de l'ajout de la bannière ressemblerait à ceci :

```
Form f=new Form();
//...
Ads banniere=new Ads();
banniere.setAppID("ID DE LA PUB ICI");

f.addComponent(banniere);
//...
```

ou

```
//...
Ads banniere=new Ads("ID DE LA PUB ICI");
f.addComponent(banniere);
```

ou

```
//...
Ads banniere=new Ads("ID DE LA PUB ICI", true);
f.addComponent(banniere);
```

Pour ce dernier exemple, le constructeur a un deuxième paramètre qui est un booléen. Celui-ci permet d'actualiser l'image de la bannière toutes les 30 secondes si sa valeur est à `true`.

La classe Ads contient d'autres méthodes utiles pour vous permettre de mieux cibler les publicités à afficher. Par exemple, la méthode `setAge(String)` permet de spécifier l'âge des personnes à cibler, `setGender(String)` leur sexe, et bien d'autres méthodes du genre pour apporter plus de précisions.

Note > *Inneractive est disponible sur toutes les plateformes couvertes par Codename One.*

1.3. Vserv

Vserv est également un leader dans le domaine de la publicité mobile, bien implanté sur les marchés émergents et anglophones, qui propose d'insérer des publicités en mode plein écran ou sous forme de bannières. Seul son mode plein écran est implémenté en interne par Codename One. Si vous préférez utiliser les bannières, alors vous devrez le coder vous-même en utilisant l'interface native (voir [Section 2, Code natif \(au-delà des limitations\)](#)).

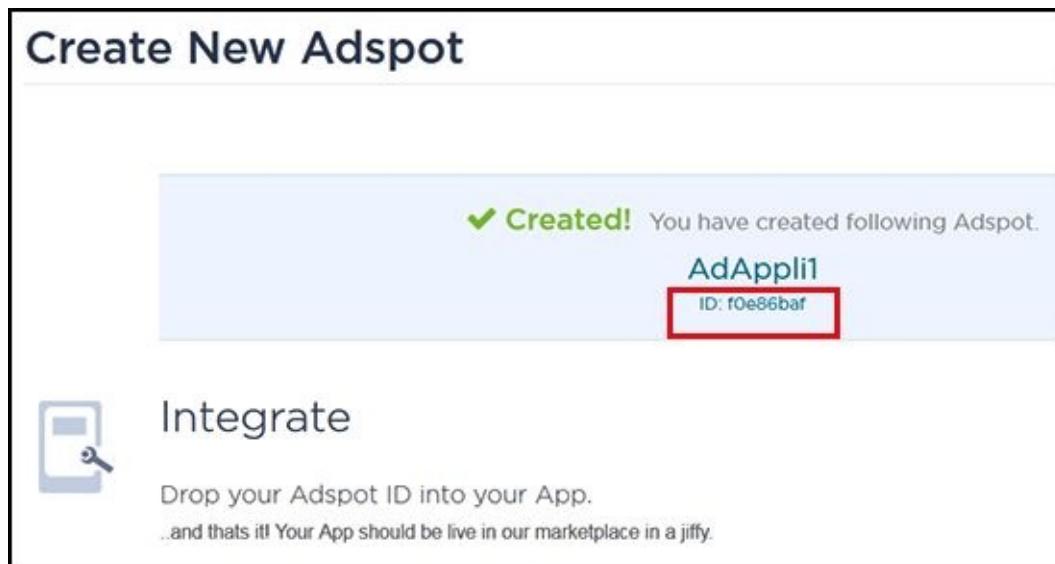
Comme avec Google AdMob, l'insertion gérée en interne est très simple et ne nécessite l'écriture d'aucune ligne de code. Vous devez juste enregistrer un identifiant fourni par la régie publicitaire et configurer l'affichage. Ces publicités s'afficheront au lancement de l'application avant l'apparition de sa première fenêtre, puis pendant les transitions d'une fenêtre à une autre en fonction d'un timing spécifié à l'avance.

Avant toute chose, rendez-vous sur le [site de Vserv](#) (en anglais), inscrivez-vous puis identifiez-vous pour avoir accès à votre espace membre. Dans ce dernier, allez dans le menu APPS & SITES et cliquez sur Adspots.

Note > *L'adspot est l'identifiant unique associé à la publicité et inséré dans l'application de votre choix.*

Sur la page qui s'affiche ensuite, cliquez sur App dans la zone Select Type puis sur New dans la zone Select Adspots pour créer le nouvel adspot. Remplissez le formulaire qui se présente à vous puis validez. Une fois l'adspot créé, copiez le code qui se trouve devant le mot ID (voir la [Figure 8.5](#)). C'est l'identifiant que nous allons enregistrer dans les propriétés de l'application.

Figure 8.5 : Créer un nouvel adspot



Dans NetBeans, cliquez droit sur le nom du projet de votre application et sélectionnez l'entrée Propriétés. Ouvrez la catégorie Monetize et vous verrez une interface semblable à celle de la [Figure 8.6](#).

Figure 8.6 : La page Monetize dans les propriétés du projet



Dans le champ Zone Id, collez l'identifiant de l'adspot que vous aviez copié précédemment. Dans le champ Transition Ad Timeout, entrez en millisecondes la durée de l'intervalle souhaité entre deux affichages de la publicité pendant les transitions d'une interface à une autre. Pour désactiver cette fonctionnalité, saisissez une valeur très élevée. Enfin, cochez la zone Full Screen pour activer l'affichage plein écran. Si cette option n'est pas cochée alors l'affichage de la bannière publicitaire se fera sous forme d'une fenêtre pop-up (qu'on peut fermer). Une fois terminé, fermez la fenêtre des propriétés. Compilez, déployez et surveillez vos revenus dans votre tableau de bord de Vserv.

Note > Les publicités avec Vserv sont géolocalisées donc ne soyez pas surpris si vous ne voyez pas de publicité quand vous lancez votre application. Cela peut simplement être dû au fait qu'aucune publicité ne couvre actuellement votre zone. Vserv est supporté par toutes les plateformes couvertes par Codename One.

Figure 8.7 : Aperçu d'une bannière Vserv



1.4. Flurry Ads

Parmi les régies publicitaires visant les plateformes mobiles, Flurry ne peut passer inaperçue. S'étant d'abord fait un nom dans la collecte des statistiques des applications mobiles, Flurry est aussi passé à la publicité depuis son rachat par la société Yahoo. Comme pour les autres, vous devez vous rendre sur [le site](#) (en anglais) pour vous inscrire, récupérer une clé (pour iOS et/ou Android), créer des publicités et récupérer leurs ID. La bannière à afficher occupera tout l'écran de l'appareil et ne sera ajoutée à aucun Form à l'intérieur de vos applications.

À la différence des trois régies précédentes, le support de Flurry Ads n'est pas intégré par défaut à l'API de Codename One. Vous devez l'ajouter à l'aide du plug-in [Flurry Ads And Analytics Support](#). Téléchargez son fichier .cn1lib et intégrez-le à votre projet comme expliqué au chapitre [Plug-ins et code natif, Section 1.2, Utilisation d'un plug-in](#). Vous pouvez maintenant passer à l'écriture du code de l'application que voici :

```
FlurryManager adsManager=new FlurryManager();
```

```
❶ adsManager.init("CLE DE L'APPLICATION RECUPEREE SUR FLURRY"); ❷  
adsManager.startSession(); ❸ adsManager.setAdSpaceName("ID DE LA PUB ICI"); ❹  
adsManager.fetchAd(); ❺ adsManager.isAdReady(); ❻ adsManager.displayAd(); ❻  
adsManager.destroyAd(); ❻
```

❶ Création d'un objet FlurryManager.

❷ Initialisation de Flurry Ads avec ajout de la clé de l'application.

❸ Démarrage d'une session.

❹ Ajout de l'ID de l'espace de publicité à afficher.

❺ Recherche de la disponibilité d'une publicité (cette recherche sera effectuée en tâche de fond).

❻ Permet de vérifier si la publicité est prête à être affichée.

❼ Affiche la publicité si elle prête.

❽ Détruit la publicité.

Note > Le même plug-in de Flurry permet aussi de récupérer les statistiques d'utilisation de vos applications. Dans ce cas, vous n'aurez besoin d'insérer que les lignes de code ❶ à ❸. Pour en savoir plus sur les statistiques, voir la [Section 4, Rapports statistiques](#).

La clé à ajouter dans la méthode `init()` de FlurryManager est différente en fonction de la plateforme que vous visez. Dans ce cas-ci, il peut s'agir d'iOS ou d'Android. Si vous voulez viser les deux plateformes avec le même code alors, effectuez la vérification

suivante :

```
String nomPlateforme=Display.getInstance().getPlatformName();  
❶ if(nomPlateforme.equals("ios")){ ❷ adsManager.init("CLE IOS DE L'APPLICATION"); } else if(nomPlateforme.equals("and")) { ❸ adsManager.init("CLE ANDROID DE L'APPLICATION"); }
```

❶ La méthode `getPlatformName()` retourne un mot de trois lettres qui correspond à la plateforme sur laquelle s'exécute l'application.

❷ Nous vérifions s'il s'agit de la plateforme iOS.

❸ Nous vérifions s'il s'agit de la plateforme Android.

L'intégration de Flurry à votre application n'est pas terminée. Vous devez encore ajouter quelques arguments de compilation (voir [Annexe 2 : Les arguments de compilation](#) pour plus d'informations) dans les propriétés du projet. Pour y accéder, cliquez droit sur votre projet, choisissez Propriétés puis allez sous l'onglet Build Hints.

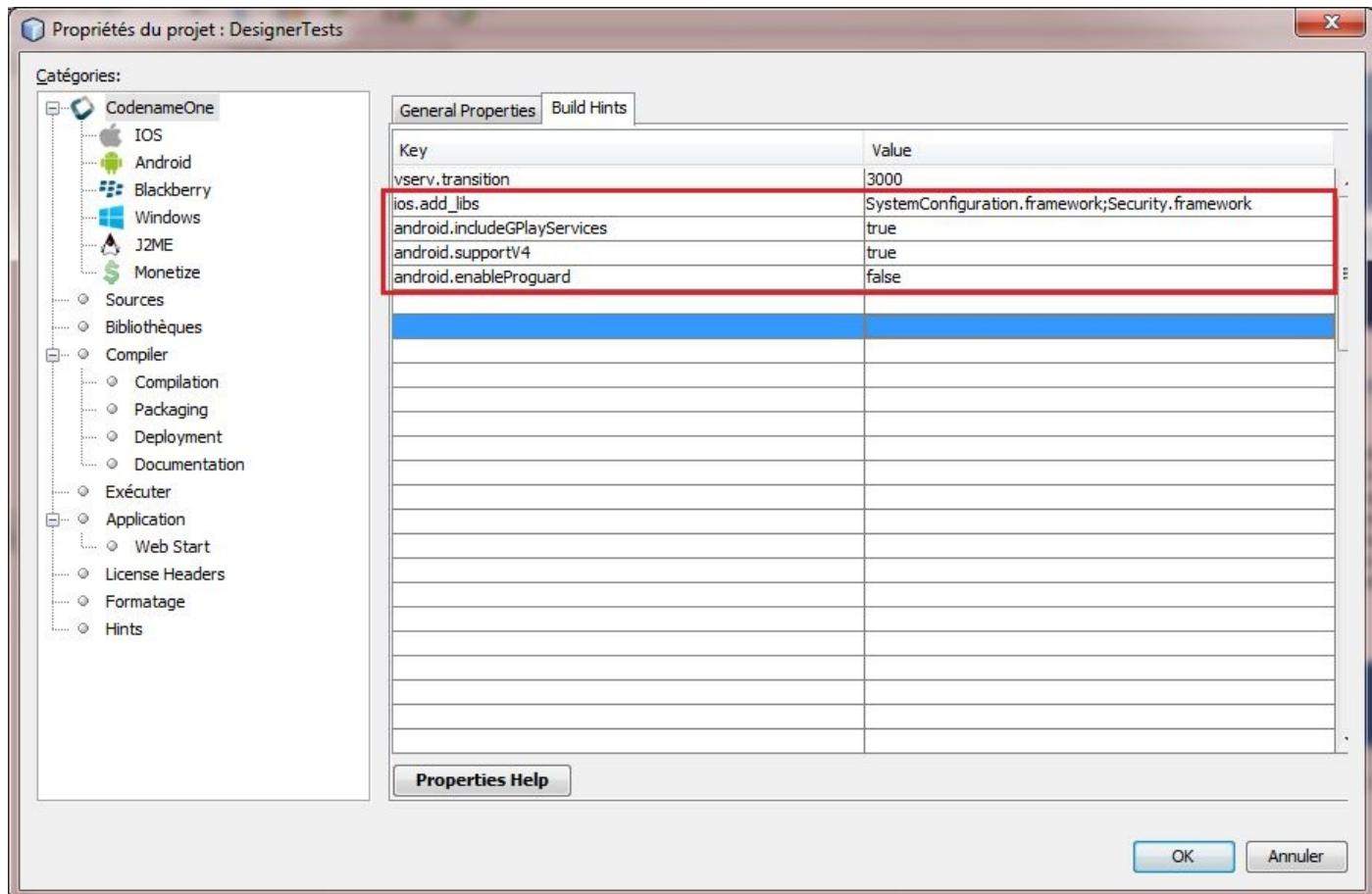
Pour iOS, ajoutez l'argument et la valeur suivants :

```
ios.add_libs = SystemConfiguration.framework;Security.framework
```

Pour Android, ajoutez les trois arguments et valeurs suivants :

- `android.includeGPlayServices = true`
- `android.supportV4 = true`
- `android.enableProguard = false`

Figure 8.8 : Arguments de compilation de Flurry Ads



Note > Sous Codename One, Flurry Ads n'est supporté que sur les plateformes iOS et Android.

2. Effectuer un paiement depuis une application

Dans cette section, nous allons voir comment installer un module de paiement et vendre des services ou produits au sein d'une application gratuite ou payante.

2.1. ZooZ

ZooZ, créé par trois anciens de Verisign, propose l'intégration d'un service de paiement à l'intérieur d'une application mobile. En échange, ZooZ prend une commission sur les montants récoltés à partir de son service. Le reste du montant est transféré par PayPal au développeur. Sous Codename One, ZooZ est disponible sous iOS et Android. Pour y recourir, vous devez d'abord vous inscrire [sur leur site](#), puis vous connecter et déclarer votre application. Comme avec les régies publicitaires, vous allez obtenir une clé qu'il faudra ensuite enregistrer dans les propriétés du projet.

Une fois connecté, ajoutez une nouvelle application, remplissez le formulaire proposé puis copiez la clé de l'application (voir la [Figure 8.9](#)). Cette clé doit être enregistrée dans les propriétés du projet, onglet Monetize/ZooZ (juste à côté de l'onglet Vserv). Collez-la dans le champ approprié selon la plateforme ciblée, Android ou iOS (voir [Figure 8.10](#)).

Figure 8.9 : Déclarer son application

The screenshot shows a progress bar at the top with four stages: 'Register App' (blue), 'Integration+ Sandbox' (green, highlighted with a red border), 'Application Details' (gray), and 'Request to go live' (gray). Below the bar, it says: 'Above is your progress bar, designed to assist you with your implementation. Green shows you the stage you are at, Blue is a stage you completed and Gray is a stage you still need to complete.' The main form area has the following fields:

- App key:** 2545f90e-3825-4fb9-8c5b-4d89f9bee5e2 (highlighted with a red border)
- Logo:** NO LOGO (with a 'Change Image' link)
- App Name:** MaSuperbeAppli
- Platform:** iOS, Android, Windows Phone (checkboxes)

Figure 8.10 : Enregister la clé fournie dans les propriétés de l'application



Une fois le paramétrage terminé dans les propriétés du projet, il vous reste à écrire le code qui intègre la fonctionnalité ZooZ à l'application. Voici le code d'un exemple qui propose à l'utilisateur de votre application de choisir un montant dans une liste déroulante et d'effectuer un don de ce montant en cliquant sur un bouton.

Exemple 8.1 : Insertion d'un module de paiement

```
Form f = new Form("Ma superbe application mobile");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

final Purchase p=Purchase.getInAppPurchase();
```

❶ Label don=new Label("Effectuer un don en euros"); ❷ final ComboBox montantDon=new ComboBox(); montantDon.addItem("5"); montantDon.addItem("10"); montantDon.addItem("20"); Button envoyer=new Button("Envoyer le don"); envoyer.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent evt) { if(p.isManualPaymentSupported()){ ❸ String montant=(String)montantDon.getSelectedItem(); ❹ p.pay(Double.parseDouble(montant), "EUR"); ❺ } else { Dialog.show("Erreur", "Paiement manuel non supporté sur cette plateforme", "Ok", null); } } }); f.addComponent(don); f.addComponent(montantDon); f.addComponent(envoyer); f.show();

❶ Création d'une instance de la classe singleton Purchase.

❷ Création d'une liste déroulante contenant les montants à donner.

❸ Appel de la méthode `isManualPaymentSupported()` pour une vérification. Cette méthode permet de savoir si la fonctionnalité de paiement manuel est disponible sur la plateforme sur laquelle s'exécute l'application. Si c'est le cas alors cette méthode retourne la valeur booléenne `true`.

❹ Récupération du montant sélectionné sous forme de chaîne de caractères dans la variable `montant`.

Appel de la méthode `pay()` qui permet d'effectuer le paiement. Cette méthode a une autre variante, mais celle que nous utilisons ici prend deux paramètres. Le premier paramètre est le montant à envoyer et est de type `double`. Pour cela, nous effectuons ici une conversion du montant récupéré sous forme de chaîne de caractères en un `double`. Le second paramètre est la devise du montant à envoyer. Cette devise correspond aux trois premières lettres du nom de la devise.

Figure 8.11 : Demande de confirmation du paiement



Une pression sur le bouton d'envoi affiche une boîte de dialogue de confirmation comme l'illustre la [Figure 8.11](#). Celle-ci n'aura pas du tout le même aspect visuel quand vous exécuterez l'application sur votre smartphone. En plus, elle vous proposera de choisir le moyen de paiement (PayPal, carte bancaire, etc.) à utiliser.

Note > L'utilisation de ZooZ nécessite une validation manuelle de leur équipe. Cette validation concerne l'enregistrement d'une application et non l'ouverture d'un compte qui est gratuite. Une fois une application ajoutée, faites une demande de validation qui peut être acceptée ou refusée. Dans le second cas, une explication vous sera donnée.

2.2. In-app purchase

In-app purchase permet de vendre des éléments ou fonctionnalités à l'intérieur d'une application gratuite ou payante. Il s'agit d'une fonctionnalité native intimement liée à

chaque plateforme mobile et fournie avec leur SDK natifs. Sous Codename One, elle est supportée sur Android et iOS.

Avant d'utiliser cette fonctionnalité, vous devez créer dans la console de votre compte développeur (sur Google Play ou sur l'App Store) ce qu'on appelle un *produit intégré* en remplissant un formulaire. Dans ce formulaire, vous allez entrer un identifiant unique nommé SKU pour ce produit intégré. Cet identifiant est choisi par vous et vous devez le noter ou le copier parce que vous allez l'utiliser dans le code de votre application.

Une fois cet enregistrement terminé, vous pouvez passer à l'écriture du code d'intégration. En voici un exemple.

Exemple 8.2 : Insertion d'une fonctionnalité d'achat dans l'application

```
Form f = new Form("Ma superbe application mobile");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

final Purchase p=Purchase.getInAppPurchase();
```

```
① Label id=new Label("Id du produit"); final TextField idProduit=new TextField();
Button envoyer=new Button("Effectuer paiement"); envoyer.addActionListener(new
ActionListener() { public void actionPerformed(ActionEvent evt) {
if(p.isManagedPaymentSupported()){ ③ p.purchase(idProduit.getText()); ④ } else {
Dialog.show("Erreur", "In app purchase non supporté sur cette plateforme", "Ok",
null); } } });
f.addComponent(id); f.addComponent(idProduit); f.addComponent(envoyer);
f.show();
```

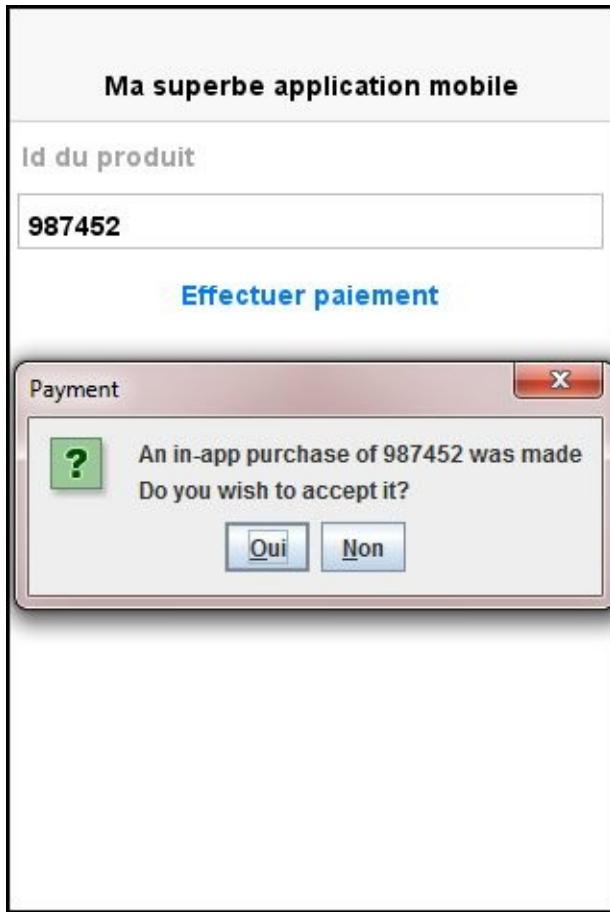
① Création d'une instance de la classe singleton Purchase.

② Création d'une zone de texte pour recueillir l'identifiant unique (SKU) du produit à acheter.

Appel de la méthode `isManagedPaymentSupported()` pour une vérification. Cette ③ méthode permet de savoir si la fonctionnalité d'in-app purchase est supportée par la plateforme. Elle retourne la valeur booléenne `true` si c'est le cas.

④ Appel de la méthode `purchase()` pour effectuer le paiement du produit dont le SKU est passé en paramètre. Le SKU ici sera le contenu de la zone de texte créée à cet effet.

Figure 8.12 : Demande de confirmation de l'achat



Encadré : À faire avant la compilation (plateforme Android)

Avant de compiler une application utilisant la fonctionnalité d'in-app purchase sous Android, procédez comme suit. Sur la fiche de votre application dans la console de Google Play, allez dans le menu Services et API puis copiez sa clé de licence (voir les encadrés de la [Figure 8.13](#)). Ouvrez la fenêtre de propriétés du projet d'un clic droit sur son nom et, sous l'onglet Build Hints, ajoutez comme clé l'instruction `android.licenseKey` et comme valeur la clé que vous avez copiée (voir la [Figure 8.14](#)). Si votre *produit intégré* est un produit consommable alors vous êtes prêt pour la compilation. Si c'est un produit non consommable alors vous devez ajouter à la fin de votre clé (une fois collée) le SKU que vous aviez choisi pour ce produit suivi du mot *nonconsume* (Exemple: `SKUXXX_nonconsume`).

Un exemple de *produit consommable* peut être par exemple l'achat de minutes de communication pour une application de voix sur IP. Un exemple de *produit non consommable* sera par exemple des fonctionnalités dans une application ou des niveaux supplémentaires dans un jeu.

Figure 8.13 : Récupération de la clé de licence de l'application

Statistiques

Notes et commentaires

Plantages et ANR

Conseils d'optimisation 2

Cloud Test Lab

Fichiers APK

Fiche Google Play Store

Catégorie de contenu

Tarifs et disponibilité

Produits intégrés à l'application

Services et API

SERVICES ET API

GOOGLE CLOUD MESSAGING (GCM)

Google Cloud Messaging (GCM) est un service qui permet d'envoyer des données de vos serveurs à vos applications. [En savoir plus](#)

Pour accéder aux statistiques GCM relatives à votre application, vous devez lui associer l'identifiant d'expéditeur GCM que vous utilisez pour celle-ci en fournissant votre clé d'API GCM.

Une fois votre application publiée, vous pouvez accéder aux statistiques GCM la concernant depuis la page des statistiques.

[Associer un identifiant d'expéditeur](#)

LICENCE ET FACTURATION VIA L'APPLICATION

Les licences vous permettent d'empêcher toute distribution non autorisée de votre application. Elles peuvent également être utilisées pour valider les achats facturés via l'application. [En savoir plus sur les licences](#)

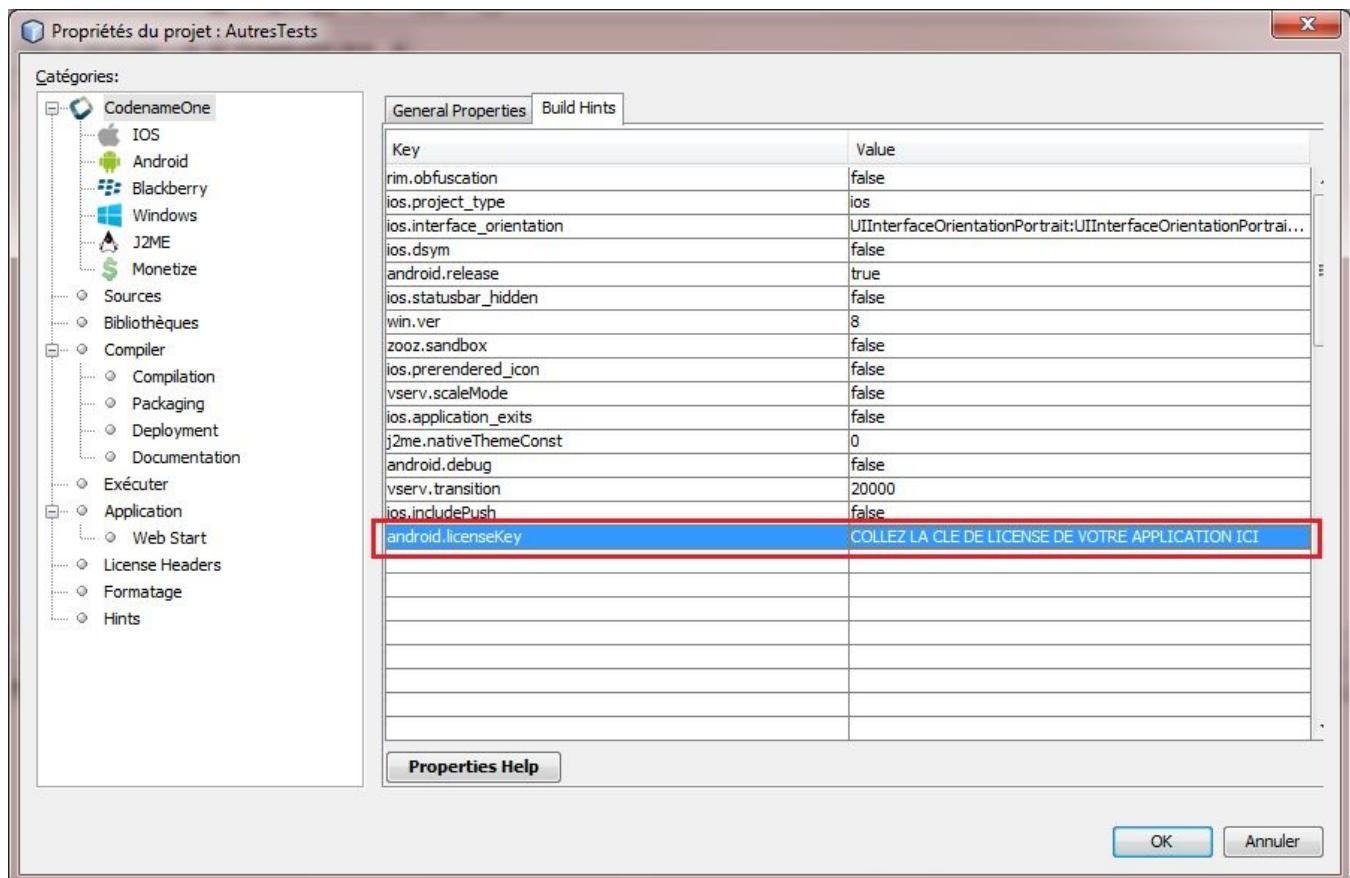
VOTRE CLÉ DE LICENCE POUR CETTE APPLICATION

Clé publique RSA codée en Base64, à inclure dans votre fichier binaire. Veuillez supprimer les espaces.

```
MIIBIjANBgkqhkiG9w0BAQEAAQABAMIIIBCgKCAQEAhidpKk0W6oHyr7CRCpmm+iWNVyoW3gyxEGb0FAySB/o3H0N7Qu/59hnv3WWKC6sgyf/ySK2bxnnfx1BKdN8NOYv/f4DE1NrqB1D37GItBHAYiD1zGBglAB1sj02QjARceP4jkPeu+z3X0to/k+nR1X48j8FNH1AgMcM410skOH4cekWY9L9eDNyicFHP8qNx8oOKCw3Iug/pjy60/Ld115sRroEOpwsN6uJERaba8EenpOFWMTC4jNwg6FY/5PONvCgBCbwXfb0Wdp9iyyLvIdaSHoFYbXubAZ7hMfuBp8QSD0JAxCW0Ompk4Cr225em2LJMMmlr+6JABTPSYfiQIDAQAB
```

SERVICES DE JEUX GOOGLE PLAY

Figure 8.14 : Insertion de la clé de licence dans le projet



Note > Sous la plateforme iOS, certaines restrictions sont liées à l'utilisation de la fonctionnalité d'in-app purchase. Vous trouverez des détails concernant ces restrictions notamment sur le [blog d'InApps](#).

2.3. Quelques méthodes de Purchase

- `boolean isRefundable(String sku)` – Indique si le remboursement du produit dont le SKU est passé en paramètre est possible. Retournera `true` si c'est le cas.
- `void refund(String sku)` – Effectue le remboursement du produit dont le SKU est passé en paramètre.
- `boolean wasPurchased(String sku)` – Indique si le produit dont le SKU est passé en paramètre est déjà acheté. Retournera `true` si c'est le cas.
- `Product[] getProducts(String[] skus)` – Retourne dans un tableau, la liste des produits dont les SKU sont passés en paramètre sous forme de tableau de chaînes de caractères. Cette méthode retourne un objet de type `Product`. Ce type représente un produit dans le catalogue d'applications de la boutique de la plateforme visée. C'est une classe constituée uniquement de getters et de setters qui fournissent ou permettent de définir les informations d'un produit comme le nom, le prix, la description et le SKU.

Fonctionnalités diverses

Dans ce chapitre, nous allons voir différentes fonctionnalités de Codename One qui n'ont pas trouvé leur place dans les chapitres précédents, mais qui sont tout aussi importantes à la conception d'une application complète et bien fournie.

1. La classe Display

Cette classe est l'une des plus importantes de Codename One. C'est elle qui gère le [thread principal](#) qui s'occupe du rendu et des événements. De plus, elle inclut de nombreuses fonctions utiles, permettant notamment d'utiliser le vibreur de l'appareil, de récupérer la dimension de l'écran, d'ouvrir une URL dans un navigateur ou encore un document quelconque, de récupérer le nom de la plateforme courante, de basculer l'application du mode portrait au mode paysage et vice versa, d'effectuer une tâche en arrière-plan et beaucoup d'autres choses encore. Je vous invite à consulter la [Javadoc](#) de cette classe pour découvrir les autres fonctionnalités qu'elle propose. Display est un singleton, donc il vous suffit de récupérer son unique instance pour l'utiliser comme c'est le cas avec la notation suivante :

```
Display.getInstance().NOM_DE_LA_METHODE_A_APPELER();
```

- `void execute(String url)` – Exécute l'élément dont le nom ou l'URL est passé en paramètre. Elle peut s'utiliser pour ouvrir un fichier de n'importe quel type (PDF, document Word, fichier audio, fichier vidéo, etc.). Il sera alors ouvert avec l'application appropriée disponible sur l'appareil. Elle s'utilise aussi pour ouvrir une adresse web dans un navigateur internet.
- `void exitApplication()` – Quitte et ferme une application.
- `Form getCurrent()` – Retourne le Form courant (Form affiché à l'écran).
- `int getDisplayWidth()` – Retourne la valeur de la largeur de l'écran.
- `int getDisplayHeight()` – Retourne la valeur de la hauteur de l'écran.
- `String getPlatformName()` – Retourne le nom de la plateforme de l'appareil. Il s'agit d'un mot de deux ou trois lettres : `ios` (pour la plateforme iOS), `and` (pour la plateforme Android), `rim` (pour BlackBerry OS), `win` (pour Windows Phone), `me` (pour J2ME).
- `boolean isEdt()` – Retourne `true` si l'action en cours se passe dans l'[EDT](#).
- `boolean isPortrait()` – Retourne `true` si le téléphone est en mode d'orientation portrait (vertical).
- `boolean isTablet()` – Retourne `true` si l'appareil est une tablette et non un téléphone.
- `void lockOrientation(boolean portrait)` – Bloque l'orientation de l'affichage dans un seul sens. Si le paramètre `portrait` est à `true` alors l'orientation sera bloquée sur le mode portrait (verticalement). Si sa valeur est `false` alors l'orientation sera bloquée sur le mode paysage (horizontalement).
- `void unlockOrientation()` – Permet de désactiver le verrouillage de l'écran dans un seul sens. En bref, elle désactive l'effet de la méthode `lockOrientation()`.
- `boolean minimizeApplication()` – Réduit et renvoie une application en arrière-plan si cette fonctionnalité est supportée par la plateforme. Elle retourne `true` si l'action

s'effectue et false dans le cas contraire.

- `void restoreMinimizedApplication()` – Permet de restaurer à l'écran l'application réduite en arrière-plan avec `minimizeApplication()`.
- `Object notifyStatusBar(String text, String titleText, String bodyText, boolean vibrate, boolean flashLights, Hashtable args)` – Crée et place une notification dans la barre de statut si cette fonctionnalité est disponible sur l'appareil. Le premier paramètre est le texte de la notification qui défilera sur la barre de statut, le deuxième est le titre de la notification, le troisième est le texte du contenu de la notification, le quatrième permet de faire vibrer l'appareil si sa valeur est à true. Le cinquième permet d'activer l'éclairage interne de l'écran du téléphone si sa valeur est à true. Le sixième et dernier permet d'ajouter des arguments divers à la notification.
- `void scheduleBackGroundTask(Runnable r)` – Permet d'exécuter une action en tâche de fond. En paramètre à cette méthode, un thread créé par Runnable. La priorité de ce thread est faible.
- `void setScreenSaverEnabled(boolean screenSaver)` – Active et désactive l'affichage de l'écran de veille de l'appareil. Le paramètre screenSaver doit être à false pour le désactiver et à true pour l'activer.
- `void vibrate(int duration)` – Fait vibrer l'appareil pendant la durée passée en paramètre. Le paramètre duration est en millisecondes.

2. Appel, e-mail, SMS

Les méthodes qui permettent d'effectuer les trois actions que nous allons voir font aussi partie de la classe Display vue à la section précédente.

- `void dial(String number)` – Ouvre l'application native de l'appareil qui permet de lancer un appel avec le numéro de téléphone passé en paramètre sous forme de chaîne de caractères.
- `void sendSMS(String number, String message)` – Envoie un SMS avec le message passé en deuxième paramètre au numéro de téléphone passé en premier paramètre.
- `void sendMessage(String[] recipients, String subject, Message msg)` – Utilise l'application d'envoi d'e-mail par défaut de l'appareil pour envoyer un e-mail aux destinataires dont les e-mails sont passés en premier paramètre. Même s'il s'agit d'un seul destinataire, il faut quand même passer un tableau de chaînes de caractères à ce premier paramètre. En deuxième paramètre, on a le sujet de l'e-mail et en troisième paramètre, le message à envoyer. Ce message doit être créé par la classe Message. Voici un exemple d'utilisation :

```
Message message=new Message("Bonjour, juste pour avoir de tes nouvelles. Tu  
me manques");  
Display.getInstance().sendMessage(new String[]{"dodericg@gmail.com"},  
"Salutation", message);
```

La méthode `sendMessage()` peut aussi être appelée directement avec la classe Message sans passer par Display parce que cette méthode est statique à l'intérieur de la classe Message.

La classe Message contient des méthodes intéressantes comme celle permettant d'attacher une pièce jointe ou celle pour définir le type MIME de l'e-mail (format texte ou HTML).

Note > *Les abonnés payants au cloud de Codename One disposent également avec `sendMessage()` d'une possibilité d'envoi d'e-mails par le cloud sans passer par les fonctionnalités natives de l'appareil. La méthode pour le faire est `sendMessageViaCloudSync()` et sa version asynchrone est `sendMessageViaCloud()`. Un message envoyé par l'une de ces deux méthodes aura pour expéditeur Codename One. C'est fait exprès pour protéger des spams.*

3. Lecture de codes-barres et de QRcodes

Les codes-barres et QRcodes sont très répandus de nos jours et il peut être pratique d'intégrer une fonctionnalité de lecture de ces codes à une application. La classe qui s'occupe de leur lecture est CodeScanner, qui est un singleton et qui utilise la caméra de l'appareil. La méthode pour lire un code-barres est scanBarcode() et celle pour lire un QRcode est scanQRCode(). Les deux méthodes ont le même prototype donc il suffit de savoir utiliser l'une pour savoir utiliser l'autre. Voici un exemple qui lit et affiche le contenu d'un QRcode dans une boîte de dialogue :

```
CodeScanner scanner=CodeScanner.getInstance();
```

```
❶ scanner.scanQRCode(new ScanResult() { public void scanCompleted(String contents, String formatName, byte[] rawBytes) { ❷ Dialog.show("Contenu du QRcode", contents, "Ok", null); } public void scanCanceled() { ❸ Dialog.show("Annulation", "Scan du code annulé", "Ok", null); } public void scanError(int errorCode, String message) { ❹ Dialog.show("Erreur", "Une erreur s'est produite. "+message, "Ok", null); } });
```

Récupération d'une instance de la classe CodeScanner ❶. Le callback ScanResult contient trois méthodes que nous avons implémentées. La première nommée scanCompleted() est appelée une fois que le scan est terminé ❷. Elle contient en premier paramètre le contenu du QRcode sous forme de chaîne de caractères. En deuxième paramètre, le format du scan et en dernier paramètre le contenu du QRcode sous forme de type byte. scanCanceled() est appelée en cas d'annulation du scan en cours ❸ et scanError() en cas d'erreur du scan ❹. Cette dernière méthode prend en premier paramètre le code d'erreur et en deuxième paramètre le message d'erreur.

4. Rapports statistiques

Lorsqu'on déploie une application, il est extrêmement précieux d'avoir des rapports sur le comportement des utilisateurs. Codename One permet d'utiliser Google Analytics, le service de tracking de sites web et d'applications de Google. Ce service fournit diverses statistiques.

C'est la classe `AnalyticsService` qui permet d'intégrer cette fonctionnalité. Avant toute chose, inscrivez-vous sur Google Analytics, déclarez le nom de domaine associé à votre application puis récupérez le code de tracking, qui sera sous la forme `UA-39018111-1`. Ne soyez pas surpris par la demande du nom de domaine, c'est parce que Google Analytics est fait à la base pour suivre des sites web. Vous pouvez utiliser le nom de domaine de votre site internet ou celui de votre application si vous en avez un. Une fois le code de tracking récupéré, ajoutez la ligne de code suivante à l'intérieur de la méthode `init()` (de préférence) ou `start()` de la classe principale de votre projet.

```
AnalyticsService.init("UA-39018111-1", "www.codenameonefr.com");
```

❶

❶ Le premier paramètre est le code que vous avez récupéré dans votre espace Google Analytics et le deuxième est l'adresse du nom de domaine que vous voulez utiliser.

C'est tout ce que vous avez à faire pour commencer à recevoir les statistiques venant de votre application. Si vous voulez avoir des rapports plus détaillés, faites appel à la méthode `visit()` sur chaque page (ou fenêtre) de votre application. Cette méthode est asynchrone.

```
AnalyticsService.visit("Identification", "Accueil");
```

❷

Dans cet exemple, la fenêtre de l'application qui est affichée à l'écran est la page ❷ d'identification (nommée ici *Identification*) et la fenêtre que nous venons de quitter est la première fenêtre de l'application (nommée ici *Accueil*).

Depuis quelque temps déjà, Google propose des rapports spécifiques pour les applications mobiles. Ce mode est désactivé par défaut dans Codename One. Pour l'activer, faites `AnalyticsService.setAppsMode(true)`. Avec ce mode activé, vous pouvez aussi recevoir des rapports sur les crashes et les erreurs qui se produiront dans vos applications. Il s'agit principalement des exceptions qui se déclencheront. La méthode pour envoyer ces rapports de crashes est `sendCrashReport()` dont le prototype est le suivant :

- `void sendCrashReport(Throwable t, String message, boolean fatal)` – Le premier paramètre est l'exception qui s'est déclenchée, le deuxième est le message d'erreur qui est limité à 150 caractères et le troisième est un booléen pour préciser si l'exception est fatale ou non à la suite de l'exécution de l'application.

5. Logging

Codename One permet de sauvegarder des informations de suivi ou d'erreur dans un fichier journal (aussi appelé *fichier log*). Cela simplifie le débogage des applications. Les informations d'un fichier journal sont sauvegardées par défaut dans un Storage. La classe qui gère ces fichiers est Log qui fournit des méthodes statiques pour simplifier ce processus.

L'écriture dans un fichier journal peut se faire avec l'une ou l'autre de ces méthodes : Log.p(String message) ou Log.e(Throwable t). La première est une méthode surchargée et prend en paramètre un texte de votre choix qui est généralement le message d'erreur à sauvegarder. La seconde prend une exception en paramètre. Ces deux méthodes n'enregistrent pas seulement les messages d'erreur dans un Storage mais les affichent aussi dans la console de l'environnement de développement.

Le contenu d'un fichier journal peut aussi être affiché dans l'application à l'aide de la méthode Log.showLog(). Celle-ci crée un Form avec un TextArea contenant les messages d'erreur enregistrés.

Voici deux exemples d'enregistrement de messages dans un fichier journal.

```
Log.p("Erreur de lecture du fichier");
```

ou

```
try {
    // Code ici
} catch(IOException ex) {
    Log.e(ex);
}
```

Vous pouvez préciser le niveau de détails que vous voulez avoir dans le fichier journal à l'aide de la méthode setLevel(int). La valeur qu'elle prend en paramètre indique le niveau de reporting à utiliser. Les valeurs possibles, de la plus basse à la plus élevée, sont : Log.DEBUG, Log.INFO, Log.WARNING, Log.ERROR. Plus le niveau est bas, moins il y aura de détails dans le message d'erreur enregistré. Outre la méthode setLevel(), vous pouvez aussi utiliser une variante de Log.p() qui prend en deuxième paramètre le niveau de reporting. Voici deux exemples :

```
Log.setLevel(Log.DEBUG);
```

ou

```
Log.p("Message d'erreur", Log.ERROR);
```

Encadré : Envoi des journaux par e-mails

Les abonnés payants de Codename One peuvent recevoir par e-mail le contenu du fichier journal. Pour cela, il faut utiliser la méthode Log.sendLog(). L'adresse e-mail du destinataire sera celle que vous utilisez comme identifiant de connexion sur le site de Codename One.

Au lieu d'envoyer ces informations manuellement avec `sendLog()`, on peut demander à l'application de le faire automatiquement en fonction des changements qui seront effectués dans le fichier journal ou en fonction des exceptions qui seront déclenchées. Le code suivant enverra toutes les cinq minutes les informations du fichier journal vers l'adresse e-mail du développeur. Ce code peut être placé dans la méthode `init(Object)` de votre projet.

```
Log.setReportingLevel(Log.REPORTING_DEBUG);
```

❶ `DefaultCrashReporter.init(true, 5);` ❷

❶ `setReportingLevel()` est semblable à `setLevel()` mais est utilisée uniquement pour les rapports envoyés par e-mail via le cloud. Les valeurs possibles sont : `REPORTING_DEBUG`, `REPORTING_NONE`, `REPORTING_PRODUCTION`.

La méthode `init()` de `DefaultCrashReporter` prend en premier paramètre un booléen pour indiquer si le rapport doit être envoyé ou non au développeur par e-mail. Mettez la valeur `true` pour l'activer.

Attention > *La fonctionnalité d'envoi des rapports du fichier journal par e-mail est réservée aux utilisateurs du compte PRO.*

6. Menu hamburger

Le menu hamburger (encore appelé *side menu*) est un style de menu popularisé par Facebook à travers son application mobile. Il s'agit d'un menu qui peut se placer sur le côté gauche ou droit de l'écran. Il est représenté par une icône constituée de trois barres

horizontales  qui fait penser à un hamburger et qui coulisse sur le côté pour laisser voir le contenu du menu. Ce style de menu est utilisé par beaucoup d'applications et leur donne un aspect visuel particulier.

6.1. Crédit d'un menu hamburger

Pour créer ce style de menu, insérez la ligne de code suivante dans la méthode `init()` ou `start()` de votre projet d'application. Peu importe que ce soit dans l'une ou dans l'autre :

```
UIManager.getInstance().getLookAndFeel().setMenuBarClass(SideMenuBar.class);
```

Cette ligne de code modifie le style du menu. Une fois ajoutée, créez vos menus comme d'habitude avec la classe `Command` et ajoutez-les à vos `Form` avec `addCommand()`. Voici un exemple de création d'un menu hamburger.

Exemple 9.1 : Crédit d'un menu hamburger

```
UIManager.getInstance().getLookAndFeel().setMenuBarClass(SideMenuBar.class);

Form f=new Form("Menu hamburger");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

Command about=new Command("A propos");
Command quitter=new Command("Quitter");

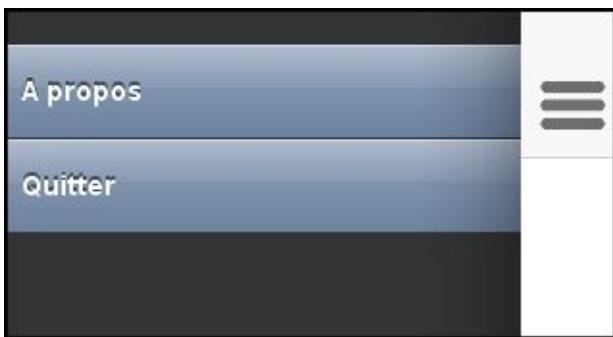
f.addCommand(about);
f.addCommand(quitter);

f.show();
```

Figure 9.1 : Menu hamburger fermé



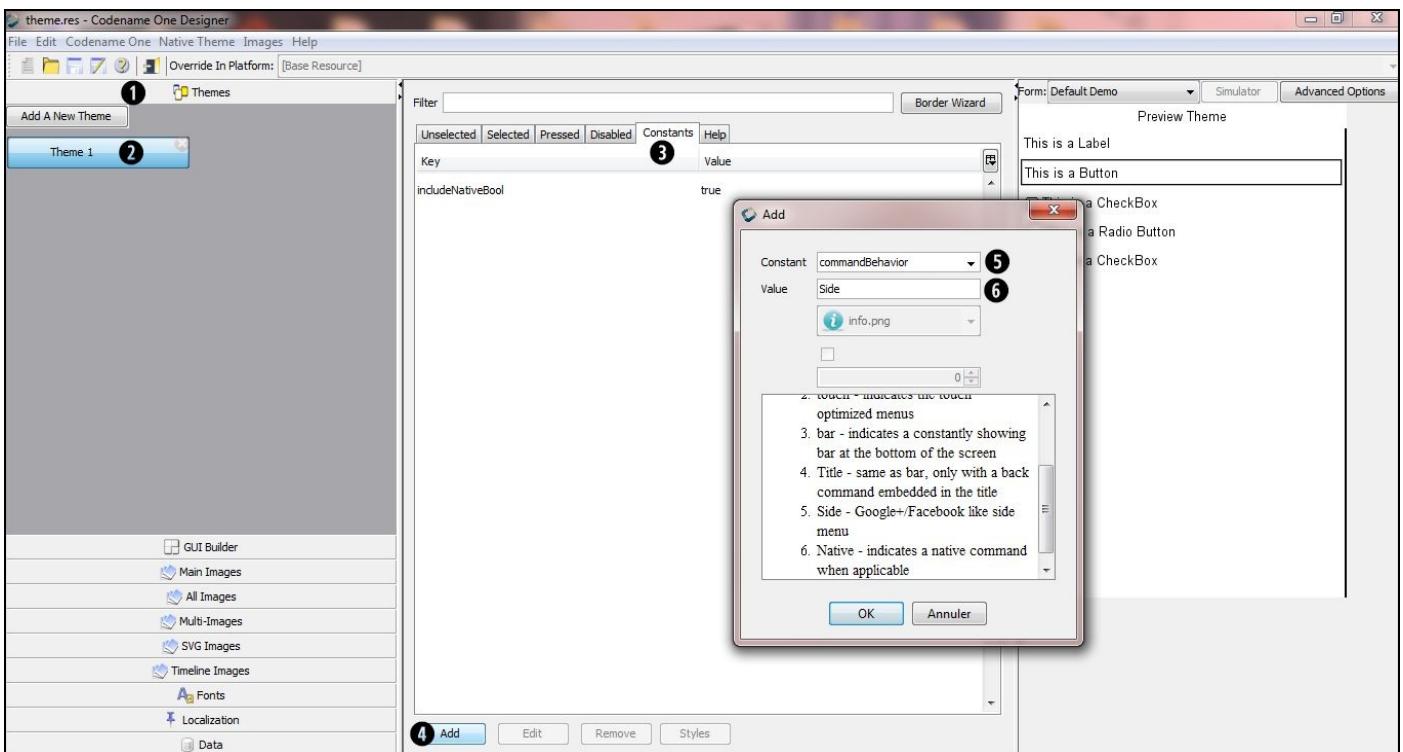
Figure 9.2 : Menu hamburger ouvert



L'autre méthode pour aboutir au même résultat est de déclarer une [constante de thème](#) dans le fichier de ressources avec Codename One Designer (voir Figure 9.3). Ouvrez le fichier theme.res de votre projet et sous la catégorie Themes **1**, sélectionnez votre thème **2** puis allez sous l'onglet Constants **3**. Cliquez sur le bouton Add **4** en dessous de la fenêtre.

Dans la zone Constant de la fenêtre qui s'affichera, sélectionnez *commandBehavior* **5** puis dans la zone Value, écrivez le mot *Side* **6**. C'est terminé ! Validez pour fermer la fenêtre et enregistrez les modifications effectuées.

Figure 9.3 : Ajout de la constante de thème *commandBehavior*



6.2. Modification de l'icône par défaut

Pour changer l'icône par défaut du menu hamburger (avec ses trois traits horizontaux), [ajoutez d'abord l'image à votre fichier de ressources](#), puis retournez sous l'onglet Constants de votre thème et ajoutez une autre constante à votre thème comme sur la

Figure 9.4. Sélectionnez votre icône dans la liste déroulante, validez puis enregistrez les modifications. Exécutez pour voir le résultat (voir [Figure 9.5](#)).

Note > Pour l'icône, utilisez une [multi-image](#) pour pouvoir supporter différentes résolutions d'écran.

Figure 9.4 : Ajout de la constante de thème sideMenuItemImage

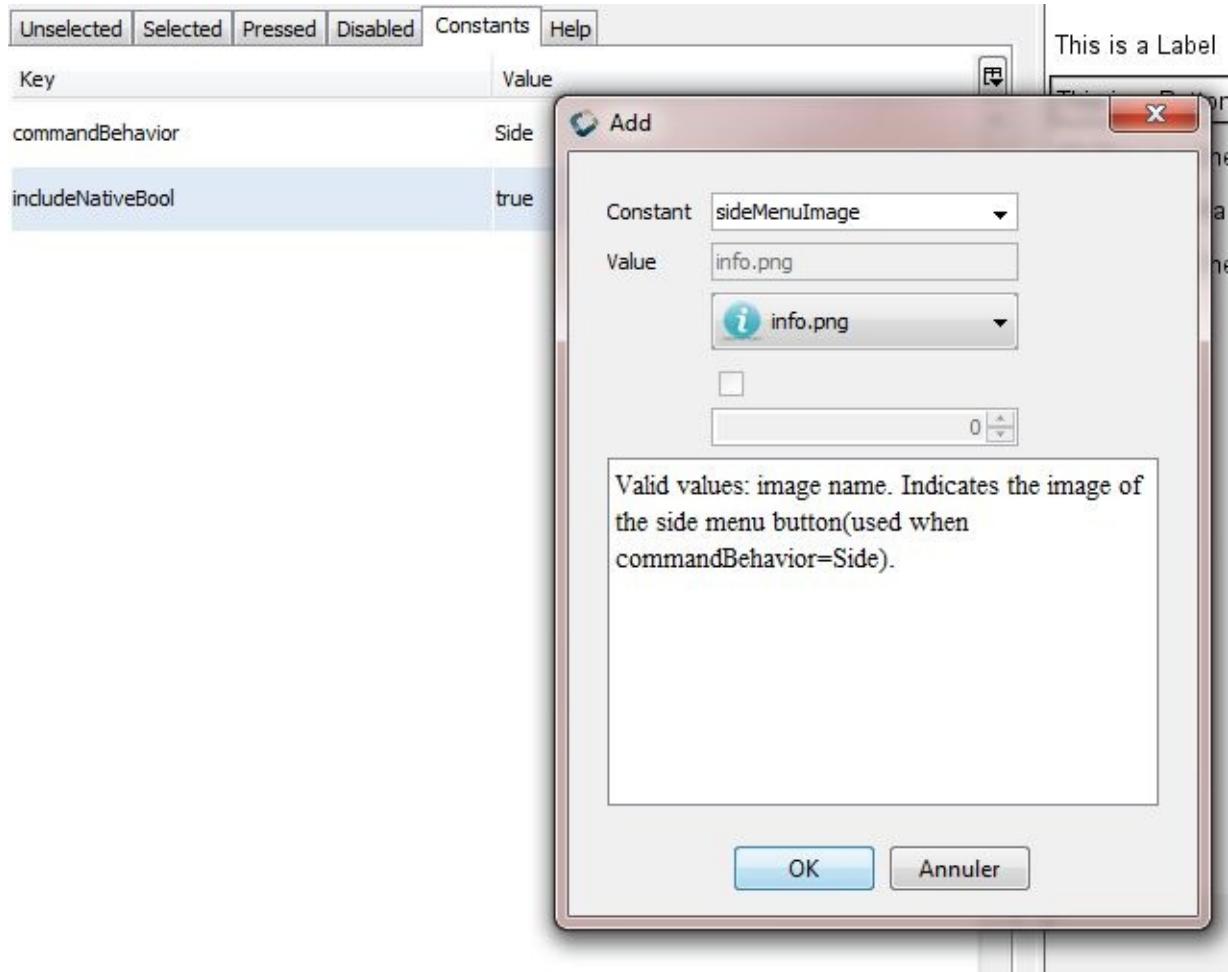


Figure 9.5 : Menu hamburger avec icône personnalisée



6.3. Changement de l'emplacement du menu

Par défaut, le menu hamburger se place sur la gauche mais il est possible de le placer à droite. Mieux, vous pouvez aussi en placer un de chaque côté. Dans l'exemple suivant,

nous allons déplacer la Command A propos (représentée par l'objet about) du menu de gauche pour l'ajouter à un autre menu qui sera créé sur la droite. Le code qui effectue ce déplacement et qui crée aussi automatiquement le menu de droite est le suivant :

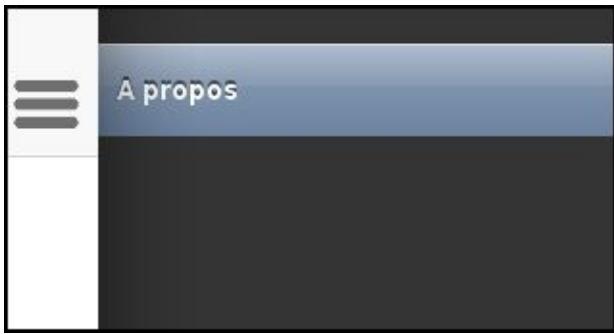
```
about.putClientProperty(SideMenuBar.COMMAND_PLACEMENT_KEY,  
SideMenuBar.COMMAND_PLACEMENT_VALUE_RIGHT);
```

Dans cet exemple, nous déplaçons la Command A Propos (représentée par l'objet about) sur le menu hamburger qui sera créé à droite. Pour cela, nous utilisons la méthode `putClientProperty()` qui prend en premier paramètre la clé `COMMAND_PLACEMENT_KEY`, lequel définit l'emplacement d'un objet Command, et en deuxième paramètre la valeur `COMMAND_PLACEMENT_VALUE_RIGHT` pour préciser que la Command about doit être placée dans le menu de droite. Le résultat est celui des [Figure 9.6](#) et [Figure 9.7](#). Les valeurs possibles de ce deuxième paramètre sont : `COMMAND_PLACEMENT_VALUE_RIGHT` (positionnement à droite), `COMMAND_PLACEMENT_VALUE_LEFT` (positionnement à gauche), `COMMAND_PLACEMENT_VALUE_TOP` (positionnement au-dessus, sur la barre de titre).

Figure 9.6 : Menu hamburger à gauche et à droite



Figure 9.7 : Ouverture du menu hamburger de droite



6.4. Ajout d'un composant quelconque

Il est possible d'ajouter n'importe quel composant à l'intérieur de l'espace d'un menu hamburger. Pour cela, utilisez la propriété `SideComponent`, ce qui aura pour conséquence de ne plus afficher le titre de la Command passé en paramètre au constructeur. Le mieux est donc de ne rien mettre comme titre à cet endroit. Voici un exemple qui ajoute un bouton à l'intérieur d'un menu hamburger :

```

Command autres=new Command(null);
autres.putClientProperty("SideComponent", new Button("Télécharger"));

```

Ce serait dommage d'avoir la possibilité d'ajouter un composant à cet endroit et de ne pas en mettre plusieurs. Voici alors un second exemple qui ajoute un bouton et une image à un menu hamburger. Ces composants seront placés dans un Container et ce dernier sera passé comme valeur de la propriété SideComponent.

Exemple 9.2 : Ajout d'un composant au menu hamburger

```

Form f=new Form("Menu hamburger");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

Image img=null;
try {
    img=Image.createImage("/bonhomme.jpg");
} catch(IOException ex){
    Dialog.show("", ex.getMessage(), "Ok", null);
}

Command about=new Command("A propos");
Command quitter=new Command("Quitter");
Command autres=new Command(null);

Container c=new Container(new BoxLayout(BoxLayout.Y_AXIS));
Label l=new Label(img);
l.getStyle().setAlignment(Component.CENTER);
Button b=new Button("Dis bonjour");
b.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        Dialog.show("", "Hello Patricia", "Ok", null);
    }
});
c.addComponent(l);
c.addComponent(b);

autres.putClientProperty("SideComponent", c);

f.addCommand(about);
f.addCommand(quitter);
f.addCommand(autres);
f.show();

```

Figure 9.8 : Image et bouton à l'intérieur d'un menu hamburger



6.5. Personnalisation de la couleur ou de l'image de fond

Pour personnaliser l'arrière-plan du menu hamburger, ajoutez à votre thème l'UIID nommé *SideNavigationViewPanel* et modifiez toutes les options que vous voulez avec le Codename One Designer (voir la [Section 4, Les thèmes](#) pour plus d'informations sur le rôle et l'utilisation d'un UIID sur un composant).

Si votre intention est de modifier plutôt l'apparence des Command alors, utilisez l'UIID *SideCommand*.

6.6. Les constantes de thèmes du menu hamburger (side menu)

- *sideMenuImage* – Remplace l'icône (voir la [Section 6.2, Modification de l'icône par défaut](#)).
- *sideMenuPressImage* – Définit une image qui s'affichera quand on appuiera sur le menu.

- *rightSideMenuImage* – Identique à *sideMenuImage* mais s’applique au menu placé à droite.
- *rightSideMenuPressImage* – Identique à *sideMenuPressImage*, mais s’applique au menu placé à droite.
- *sideMenuShadowImage* – Définit l’ombrage présent sur le bord du menu.
- *hideLeftSideMenuBool* – Masque le menu placé à gauche de l’écran.
- *hideBackCommandBool* – Masque le menu Back (Retour) ajouté automatiquement à l’interface quand on quitte un Form pour un autre. Ceci concerne ceux qui utilisent le GUI Builder (et non du code) pour la création des Form.
- *sideMenuFoldedSwipeBool* – Par défaut, un menu hamburger peut s’ouvrir quand on effectue un effet de glissement avec le doigt à l’écran. Pour désactiver cette possibilité, passez la valeur de cette constante à `false`.

7. Tirer et relâcher pour actualiser

Communément appelée en anglais *Pull to refresh*, cette fonctionnalité est un concept qui a été popularisé par le site de microblogging Twitter à travers son application mobile officielle. Ça consiste à tirer vers le bas et ensuite relâcher la surface d'un Container (ou d'un Form) pour actualiser son contenu ou recevoir tout simplement des mises à jour. La méthode qui s'occupe de ça se nomme `addPullToRefresh()` et elle prend en paramètre un objet de type `Runnable` qui permet d'exécuter l'action à effectuer dans un thread parallèle à l'`EDT`. Voici un exemple d'utilisation de cette méthode qui nous permettra de créer automatiquement des boutons et de les ajouter à la fenêtre à chaque fois que nous allons tirer et relâcher la surface du Form.

Exemple 9.3 : Création de boutons à chaque exécution de pull to refresh

```
final Form f = new Form("Ma superbe application mobile");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

f.getContentPane().addPullToRefresh(new Runnable() {

    public void run() {
        //---Action à effectuer une fois la surface relachée---

        Button b=new Button("Hello, je suis un nouveau bouton");
        f.addComponent(b);
    }
});

f.show();
```

Figure 9.9 : Aperçu du fonctionnement de pull to refresh (vidéo)



Encadré : Bon à savoir

Ne vous souciez pas du fait que le texte Release to refresh... soit en anglais. Il vous suffit d'appliquer ce que nous avons vu à la [Section 7, Internationalisation/localisation](#) pour traduire ce texte en français ou dans la langue de votre choix. Comme clés ou constantes de traduction, vous pouvez utiliser `pull.release` qui a pour valeur par défaut le texte Release to refresh....

Si vous avez envie de modifier l'apparence du texte (couleur, taille, décoration, etc.) et tout ce qui a trait au visuel, ouvrez le fichier `theme.res` de votre projet et ajoutez à votre thème l'[UIID PullToRefresh](#) que vous pouvez personnaliser à volonté (voir la [Section 4, Les thèmes](#) sur le rôle et l'utilisation de l'UIID sur un composant).

8. Lecture d'un fichier CSV

Le format CSV est simple à lire et à manipuler. Ses données sont constituées uniquement de contenus séparés les uns des autres par une virgule ou un point virgule. Vous pouvez par exemple vous retrouver face à un service web qui retourne ce format de données. Dans ce cas, vous aurez besoin de la classe CSVParser. Nous allons écrire un exemple qui va nous permettre de lire les contenus d'un fichier CSV et de les afficher sur une interface avec des MultiButton. Pour cela, créez un fichier `livres.csv` que vous allez déposer dans le dossier SRC de votre projet et qui aura le contenu suivant :

```
Codename One;Eric Dodji Gbofu;2015;  
Neo4j;Sylvain Roussy;2015;  
Qt;Jonathan Courtois;2013
```

Le code de lecture et d'affichage est le suivant :

Exemple 9.4 : Lecture de données au format CSV

```
Form f=new Form("Lecture fichier CSV");  
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));  
  
InputStream stream=Display.getInstance().getResourceAsStream(getClass(),  
"/livres.csv");  
  
❶ CSVParser parser=new CSVParser(';',  
contenus=parser.parse(stream); ❷ try{ String[][]  
int nbreLigne=contenus.length; ❸ for(int  
i=0;i<nbreLigne;i++){ ❹ MultiButton mb=new MultiButton(); mb.setTextLine1("Livre:  
"+contenus[i][0]); mb.setTextLine2("Auteur: "+contenus[i][1]);  
mb.setTextLine3("Année: "+contenus[i][2]); f.addComponent(mb); } } catch  
(IOException ex){ Dialog.show("Erreur", ex.getMessage(), "Ok", null); } stream.close();  
f.show();
```

❶ Chargement du contenu du fichier `livres.csv` sous forme de `InputStream`.

Création d'un objet CSVParser. Nous passons en paramètre le caractère point-virgule
❷ `(';')` pour préciser que c'est le séparateur que nous allons utiliser. Si aucun paramètre
n'est spécifié alors le caractère virgule `(,)` sera utilisé par le parser.

❸ Lecture du contenu du fichier CSV avec la méthode `parse()` qui prend en paramètre le
stream du fichier. Elle retourne un tableau de chaînes de caractères à deux dimensions.

❹ Récupération du nombre de lignes lues dans le fichier.

Utilisation d'une boucle pour la création d'un MultiButton pour chaque ligne du
fichier. Le contenu de chaque MultiButton sera le nom d'un livre, le nom de son auteur
et l'année de publication.

Figure 9.10 : Affichage des données du fichier CSV

Lecture fichier CSV
Livre: Codename One Auteur: Eric Dodji Gbofu Année: 2015
Livre: Neo4J Auteur: Sylvain Roussy Année: 2015
Livre: Qt Auteur: Jonathan Courtois Année: 2013

9. Lecture d'un fichier XML

Pour la lecture du format XML, nous allons prendre le même exemple que celui du fichier CSV. Le résultat final sera donc le même, mais cette fois avec des données lues depuis un fichier XML. Nous allons voir deux cas. L'un avec du contenu XML qui contient des attributs et un autre sans attributs.

Le contenu XML (fichier `livres.xml` placé dans le dossier `SRC` du projet) utilisant des attributs dans les balises est le suivant :

```
<?xml version="1.0">
<livres>
<livre nom_ouvrage="Codename One" auteur="Eric Dodji Gbofu" annee="2015">
</livre>
<livre nom_ouvrage="Neo4j" auteur="Sylvain Roussy" annee="2015"></livre>
<livre nom_ouvrage="Qt" auteur="Jonathan Courtois" annee="2013"></livre>
</livres>
```

Voici le code Java qui nous permettra de lire ce fichier et d'afficher son contenu dans des MultiButton comme celui de l'exemple du CSVParser.

Exemple 9.5 : Lecture d'un fichier XML contenant des attributs dans les balises

```
Form f=new Form("Lecture fichier XML");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

InputStream stream=Display.getInstance().getResourceAsStream(getClass(),
"/livres.xml");
```

❶ XMLParser parser=new XMLParser(); ❷ Element racine=parser.parse(new InputStreamReader(stream)); ❸ int nbreEnfants=racine.getNumChildren(); ❹ for(int i=0;i<nbreEnfants;i++){ ❺ Element elmt=racine.getChildAt(i); MultiButton mb=new MultiButton(); mb.setTextLine1("Livre: "+elmt.getAttribute("nom_ouvrage")); mb.setTextLine2("Auteur: "+elmt.getAttribute("auteur")); mb.setTextLine3("Année: "+elmt.getAttribute("annee")); f.addComponent(mb); } stream.close(); f.show();

❶ Chargement du contenu du fichier `livres.xml` sous forme de flux `InputStream`.

❷ Création d'un objet `XMLParser`.

Lecture du contenu de `livres.xml` avec la méthode `parse()` à travers le flux. Cette ❸ méthode retourne un objet de type `Element` qui contient les données à l'intérieur de la balise racine `<livres>`.

❹ Récupération du nombre de balises enfants de la balise racine `<livres>`.

Création d'une boucle pour récupérer toutes les balises enfants de la balise racine. À la suite de cette récupération, un `MultiButton` est créé pour chacune de ces balises et le

- ❸ contenu de chaque ligne du MultiButton est la valeur des attributs de chaque balise enfant.

Remplaçons maintenant le contenu du fichier `livres.xml` par le code suivant qui n'utilise plus d'attributs.

```
<?xml version="1.0">
<livres>
<livre>
<nom_ouvrage>Codename One</nom_ouvrage>
<auteur>Eric Dodji Gbofu</auteur>
<annee>2015</annee>
</livre>

<livre>
<nom_ouvrage>Neo4j</nom_ouvrage>
<auteur>Sylvain Roussy</auteur>
<annee>2015</annee>
</livre>

<livre>
<nom_ouvrage>Qt</nom_ouvrage>
<auteur>Jonathan Courtois</auteur>
<annee>2013</annee>
</livre>
</livres>
```

Le code de lecture de ce nouveau contenu nous donnerait ceci :

Exemple 9.6 : Lecture d'un fichier XML ne contenant que des balises sans attributs

```
Form f=new Form("Lecture fichier XML");
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));

InputStream stream=Display.getInstance().getResourceAsStream(getClass(),
"/livres.xml");
XMLParser parser=new XMLParser();

Element racine=parser.parse(new InputStreamReader(stream));
int nbreEnfants=racine.getNumChildren();

for(int i=0;i<nbreEnfants;i++){
    Element tagLivre=racine.getChildAt(i);

❶ Element tagNumOuvrage=tagLivre.getChildAt(0); ❷ Element
tagAuteur=tagLivre.getChildAt(1); ❸ Element tagAnnee=tagLivre.getChildAt(2); ❹ ❺
MultiButton mb=new MultiButton(); mb.setTextLine1("Livre:
"+tagNumOuvrage.getChildAt(0).getText()); mb.setTextLine2("Auteur:
"+tagAuteur.getChildAt(0).getText()); mb.setTextLine3("Année:
"+tagAnnee.getChildAt(0).getText()); f.addComponent(mb); } f.show();
```

Après avoir lancé une boucle sur le nombre de balises enfants à l'intérieur de la balise

❶ racine, nous récupérons le contenu de la balise <livre>.

❷ Récupération de la balise <nom_ouvrage>.

❸ Récupération de la balise <auteur>.

❹ Récupération de la balise <annee>.

Création d'un MultiButton, récupération du texte à l'intérieur des balises

❺ <nom_ouvrage>, <auteur>, <annee> et ajout de ces textes sur les différentes lignes des MultiButton.

10. Cration et accs aux contacts

Dans cette section, nous allons voir comment ajouter un nouveau contact au repertoire du telephone et comment lire les contacts qui y sont enregistres. La gestion des contacts se fait a l'aide de deux classes. La premire se nomme ContactsManager et se base sur le gestionnaire de contacts de l'appareil pour ajouter ou retirer des contacts. La deuxime se nomme Contact et represente les informations relatives a un seul contact.

10.1. Ajout d'un nouveau contact

L'ajout d'un nouveau contact se fait avec la methode statique createContact() de ContactsManager. Elle retourne une chane de caracteres qui represente l'ID du contact enregistre si l'enregistrement s'est bien pass. En cas d'echec, elle retournera null. Cet ID est un identifiant unique propre a chaque contact dans le repertoire telephonique.

Voici un exemple d'utilisation de cette methode qui va enregistrer un prénom, un nom et un numero de telephone mobile dans le repertoire de contacts du telephone. Si l'enregistrement echoue alors l'utilisateur sera inform de l'echec de l'enregistrement par une bote de dialogue.

```
String idContact=ContactsManager.createContact("Bill", "Gates", null, null,  
"12121222", null);
```

❶ if(idContact==null){ Dialog.show("Erreur", "Contact non enregistr", "Ok", id); }

Enregistrement du contact dans le repertoire de contacts. Le premier paramtre de createContact() est le prénom du contact, le deuxime paramtre est son nom de famille, le troisime paramtre est son numero de telephone professionnel, le quatrime est le numero de telephone de son domicile, le cinquime est son numero de telephone mobile et le dernier paramtre est son e-mail. Dans cet exemple, nous n'insrons que le prénom, le nom et le numero de telephone mobile.

10.2. Lecture d'un ou de plusieurs contacts

Nous allons crire un exemple qui va extraire tous les contacts du repertoire de l'appareil et afficher leur nom et leur numero de telephone mobile. Lors de son execution avec le simulateur, seuls cinq contacts prednis dans l'API de Codename One seront affichs. Pour un test rel, compilez l'exemple et executez-le sur votre appareil.

Exemple 9.7 : Lecture des informations des contacts du repertoire

```
final Form f=new Form("Contacts");  
f.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
```

```

String[] contactIds=ContactsManager.getAllContactswithNumbers();

❶ for(int i=0;i<contactIds.length;i++){ Contact
c=ContactsManager.getContactById(contactIds[i]); ❷ Label labelNom=new Label("Nom:
"+c.getDisplayName()); ❸ Label labelNum=new Label("Mobile:
"+c.getPhoneNumbers().get("mobile")); ❹ Label sep=new Label("—");
f.addComponent(labelNom); f.addComponent(labelNum); f.addComponent(sep); }
f.show();

```

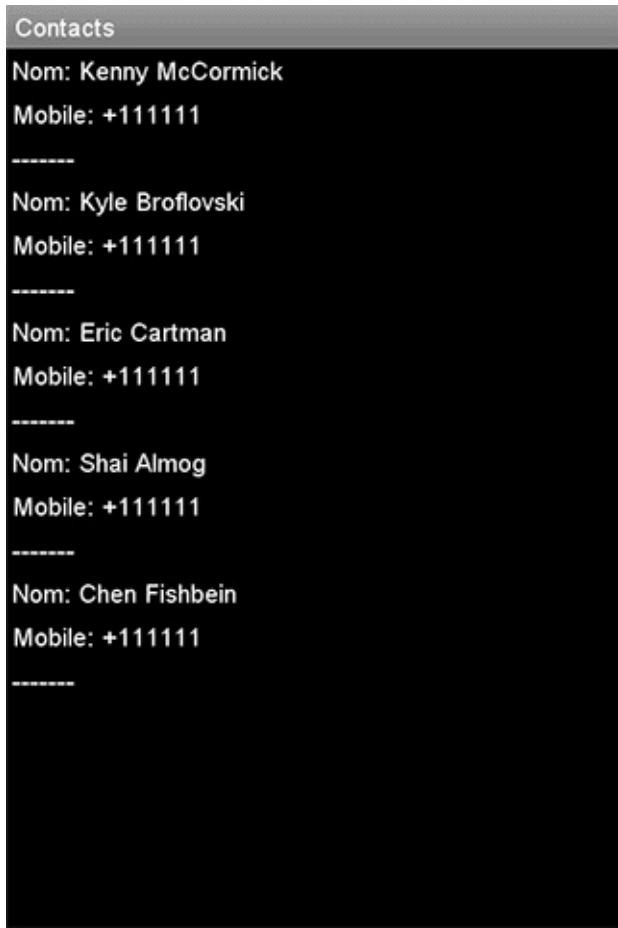
❶ `getAllContactswithNumbers()` retourne dans un tableau de chaînes de caractères les ID de tous les contacts qui ont été enregistrés avec un numéro de téléphone.

❷ `getContactById()` retourne un objet de type `Contact` correspondant à l'ID du contact passé en paramètre. Cet objet contient les informations sur le contact à savoir son prénom, son nom, son numéro, son e-mail, sa photo et bien d'autres que vous découvrirez dans la classe `Contact`.

❸ Utilisation de la méthode `getDisplayName()` de l'objet `Contact` créé précédemment pour récupérer dans un `Label` le nom du contact .

Utilisation de la méthode `getPhoneNumbers()` de l'objet `Contact` pour récupérer dans un `Label` le numéro de téléphone mobile du contact. La méthode `getPhoneNumbers()` ❹ retourne un `Hashtable` qui contient cinq numéros sous forme de clés et de valeurs. Les deux sont des chaînes de caractères. Les noms des clés sont "home", "mobile", "work", "fax", "other". Ici, nous récupérons seulement celui de "mobile".

Figure 9.11 : Lecture et affichage de contacts



10.3. Quelques méthodes de ContactsManager

- `boolean deleteContact(String id)` – Supprime le contact dont l'ID est passé en paramètre.
- `String[] getAllContacts()` – Retourne dans un tableau de chaînes de caractères tous les contacts enregistrés dans l'appareil.
- `Contact getContactById(String id, boolean includesFullName, boolean includesPicture, boolean includesNumbers, boolean includesEmail, boolean includeAddress)` – Cette méthode est semblable à `getContactById()` qui ne prend que l'ID en paramètre. L'avantage de celle-ci est que vous avez la possibilité d'appliquer des filtres sur la requête. Ainsi au lieu de retirer toutes les informations d'un contact à cause du temps de chargement que ça peut prendre, vous pouvez décider de ne retirer que les informations dont vous avez besoin. Cela apporte un gain de temps considérable et une exécution plus rapide. À part l'ID qui est en premier paramètre, les autres paramètres sont des booléens qui permettent de définir les informations à extraire ou non. Mettez `true` comme valeur pour extraire une donnée que vous voulez et `false` pour l'ignorer. Ces paramètres sont dans l'ordre suivant : nom complet, photo, numéros, e-mail et adresse.

11. Notification push

Attention > La fonctionnalité de notification push est réservée aux utilisateurs du compte PRO ou supérieur.

Les notifications push permettent d'envoyer un message (au format texte la plupart du temps) à une application sur l'appareil d'un utilisateur quelconque. L'une des particularités de ces notifications est que les messages peuvent être reçus par un appareil même si l'application concernée n'est pas en cours d'exécution. Codename One supporte les notifications sous les trois plateformes iOS, Android et BlackBerry OS.

L'intégration de cette fonctionnalité à vos applications tant au niveau de la partie cliente que de la partie serveur est grandement simplifiée par Codename One. Prenons un cas où nous avons besoin de deux applications : l'une qui constitue la partie cliente (celle qui recevra les notifications) et l'autre la partie serveur (celle qui enverra le message).

Avant toute chose, vous devez vous rendre sur les sites respectifs des concepteurs de ces plateformes pour vous inscrire et récupérer des codes et des clés qui vous seront ensuite demandés.

- Pour Android : Allez sur la page de [Google Cloud Messaging](#), suivez les instructions et récupérez deux codes : l'un est l'ID du projet et l'autre est la clé d'accès au serveur.
- Pour iOS : Vous aurez besoin de deux certificats push P12 et d'un mot de passe à récupérer dans la console de votre compte développeur. L'un de ces certificats sera à utiliser pour les tests et l'autre pour la distribution. Veillez à ce que le nom du package de la classe principale de votre application soit le même que celui que vous utiliserez pour la création des fichiers P12 dans votre console développeur chez Apple.

Attention > Le fichier P12 pour effectuer le push est différent du fichier P12 à utiliser pour la compilation de vos applications iOS.

- Pour BlackBerry OS : Rendez-vous sur la page [Push Service de BlackBerry](#). Lisez les instructions et récupérez un identifiant pour l'application, une URL, un mot de passe et le numéro d'un port. Pour BlackBerry, vous aurez besoin de vous enregistrer une première fois pour la phase de test de l'application et une seconde fois pour la phase de déploiement.

11.1. Mise en place de la partie cliente

La mise en place de la réception des notifications de la partie cliente de l'application se passe en plusieurs étapes.

1. L'application doit effectuer un enregistrement local (validation de la demande par l'utilisateur) pour pouvoir être autorisée à recevoir des notifications.

2. Une fois ce premier enregistrement effectué, un autre enregistrement doit être effectué au niveau du cloud. Par défaut, Codename One enregistre dans le cloud tous les appareils (contenant l'application) qui ont franchi la première étape.
3. Pour pouvoir recevoir les notifications, la classe principale (celle qui contient la méthode `init(Object context)`, `start()`) de la partie cliente de votre application doit hériter de l'interface `PushCallback`. Cette interface contient trois méthodes qu'il faut implémenter. Ces méthodes sont appelées à chaque fois qu'une notification est reçue.

```
public class PushNotif implements PushCallback {

    public void init(Object context) {
        // code
    }

    public void start() {
        // code
    }

    public void stop() {

    }

    public void destroy() {

    }

    public void push(String value) {
```

❶ `Dialog.show("Notification reçue", value, "Ok", null); } public void registeredForPush(String deviceId) { ❷ Dialog.show("Appareil enregistré", "ID de l'appareil: "+deviceId, "Ok", null); } public void pushRegistrationError(String error, int errorCode) { ❸ Dialog.show("Erreur d'enregistrement", error, "Ok", null); } }`

❶ Cette méthode est appelée quand la notification est reçue. Elle a en paramètre le message qui a été envoyé à l'appareil.

❷ `registeredForPush()` est appelée quand l'enregistrement de l'appareil pour la réception des notifications a été bien effectué. En paramètre, l'ID de notification push de l'appareil. Vous pouvez aussi récupérer la clé de l'appareil avec `Push.getDeviceKey()`.

❸ `pushRegistrationError()` est appelée en cas d'échec de l'enregistrement de l'appareil pour la réception des notifications. En paramètre, le message d'erreur et le code de l'erreur.

L'enregistrement proprement dit dans le cloud doit se faire avec la méthode `registerPush()` de la classe `Display`. Vous pouvez appeler le code suivant à chaque fois que l'application est lancée donc placez-le au début de la méthode `start()` de votre classe principale. À ce niveau, vous aurez besoin de l'ID du projet que vous avez récupéré chez Google et dont j'ai parlé un peu plus haut. Seule la plateforme Android a besoin de ça.

Celui des autres plateformes se fera automatiquement sans aucune précision.

```
Hashtable metaData = new Hashtable();
metaData.put(com.codename1.push.Push.GOOGLE_PUSH_KEY, "METTRE L'ID ICI");
Display.getInstance().registerPush(metaData, true);
```

Astuce > Pour plus d'éclaircissement, vous pouvez voir ces différentes étapes, y compris la partie sur la récupération des clés et codes dont vous aurez besoin, dans la vidéo [Push Notification](#) (en anglais) réalisée par l'équipe de Codename One.

Côté serveur, vous pouvez concevoir une seconde application qui servira à envoyer les messages sur les appareils ayant installé la partie cliente de votre application. L'autre méthode est de créer une interface web pour le faire. Dans le premier cas, une méthode est fournie pour envoyer le message depuis une application. Dans le deuxième cas, vous avez accès à une API web que vous pouvez intégrer dans une application web pour l'envoi.

11.2. Envoi de messages depuis une application mobile

L'envoi de messages vers les appareils ayant été enregistrés pour la réception des notifications se fait avec la méthode statique `sendPushMessage()`. Il s'agit d'une méthode surchargée et l'un de ses deux prototypes ressemble à ceci :

```
Push.sendPushMessage("MESSAGE A ENVOYER", "DESTINATION", "MODE PRODUCTION  
OU MODE TEST", "CLE AUTH RECUPEREE CHEZ GOOGLE", "URL DU CERTIFICAT DE IOS",  
"MOT DE PASSE DU CERTIFICAT DE IOS", "URL PUSH DE BLACKBERRY", "L'ID DE  
L'APPLICATION DE BLACKBERRY", "LE MOT DE PASSE PUSH DE BLACKBERRY", "LE  
NUMERO DE PORT PUSH DE BLACKBERRY")
```

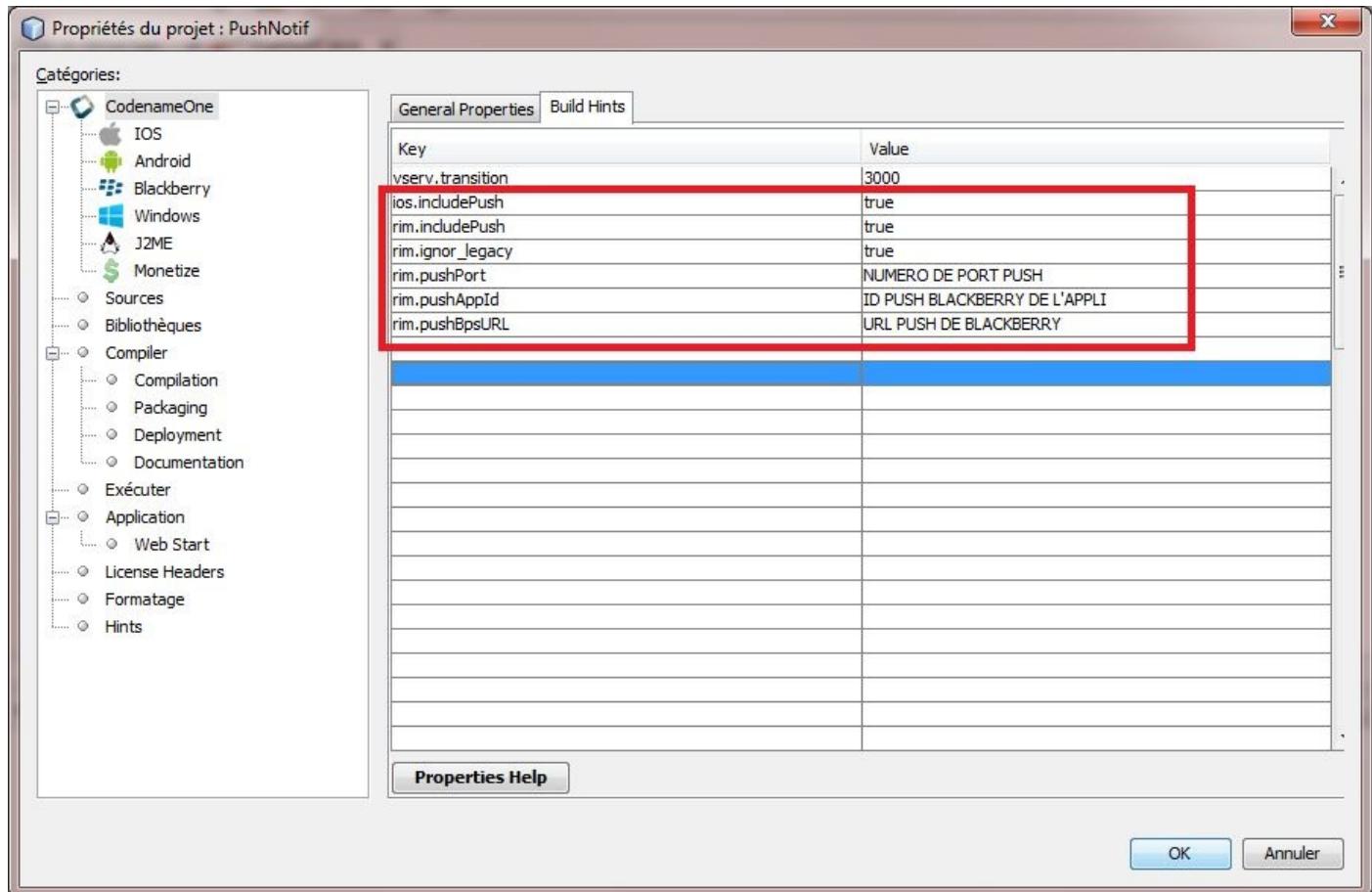
Le deuxième paramètre `DESTINATION` peut être placé à `null` si vous voulez envoyer votre message à tous les appareils qui ont votre application installée et qui ont accepté de recevoir les notifications push. Si vous visez par contre un appareil en particulier alors, mettez son ID à cet endroit. ID que vous pouvez récupérer au préalable au niveau de l'application cliente à l'aide de la méthode `Push.getDeviceKey()`.

Le troisième paramètre est un booléen pour indiquer, en ce qui concerne la plateforme iOS, si vous êtes en mode production (ou déploiement) de votre application ou en mode test. Mettez la valeur `true` si vous déployez votre application et `false` si vous êtes en mode de développement et que vous effectuez juste des tests. Les rôles des autres paramètres ont été bien résumés dans le prototype de la méthode. Là où les informations demandées concernent des plateformes que vous ne ciblez pas, mettez la valeur `null`.

L'autre version surchargée de `sendPushMessage()` ne prend pas en paramètre les quatre informations de BlackBerry OS mais seulement ceux d'Android et d'iOS. En résumé, elle prend uniquement les six premiers paramètres de celle que nous venons de voir. `sendPushMessage()` est synchrone et retourne un booléen pour informer de la bonne exécution ou non de l'envoi du message. La version asynchrone est `sendPushMessageAsync()` et a les mêmes paramètres que `sendPushMessage()`.

Pour finir, vous devez ajouter certains arguments de compilation à votre projet (voir [Annexe 2 : Les arguments de compilation](#) pour plus d'informations). Voici comment faire. Cliquez droit sur le projet et choisissez Propriétés. Sous l'onglet Build Hints, ajoutez les clés et valeurs qui sont encadrées sur la [Figure 9.12](#) en fonction des plateformes que vous visez. Ne mettez que celui de l'iOS si vous visez l'iOS, celui de BlackBerry si vous visez BlackBerry OS ou les deux. Celui d'Android n'a pas besoin d'arguments de compilation.

Figure 9.12 : Arguments de compilation de notification push



11.3. Envoi de messages depuis une application web

Dans le cas d'une application web pour l'envoi des messages, utilisez votre langage web préféré avec l'API web dédié pour l'envoi de la requête. Celle-ci doit être de type POST. L'URL à laquelle vous devrez envoyer la requête est : <https://codename-one.appspot.com/sendPushMessage>.

Le [Tableau 9.1](#) répertorie les arguments que vous pouvez passer à cette URL.

Tableau 9.1 : Arguments de l'URL de l'API web de notification push

Argument	Descriptif
device	L'ID de l'appareil qui recevra la notification. Une valeur null l'enverra à tous les appareils et non plus à un appareil en particulier.

email	L'adresse e-mail du développeur.
packageName	Le nom du package de la classe principale de votre application. Ce nom est très important pour l'iOS.
type	Il s'agit du type de notification push à envoyer. La valeur par défaut est 1. La valeur 2 indique que la notification ne sera pas affichée à l'utilisateur.
auth	La clé AUTH récupérée chez Google.
cert	L'URL vers le fichier de certificat push P12 de l'iOS.
certPassword	Le mot de passe du fichier de certificat push P12 de l'iOS.
production	Pour indiquer si vous êtes en mode production ou en mode de test. Cet argument ne concerne que la plateforme iOS et prend la valeur true ou false.
body	Le message à envoyer aux appareils.
burl	L'URL push de BlackBerry.
bbAppId	L'ID de l'application récupérée chez BlackBerry.
bbPass	Le mot de passe push récupéré chez BlackBerry.
bbPort	Le numéro de port push récupéré chez BlackBerry.

Étude de cas : Création d'une application de A à Z

Dans ce chapitre, nous allons créer une application complète avec Codename One. Nous allons donc utiliser différents éléments que nous avons vus dans les chapitres précédents. Ceux que nous utiliserons dans l'application sont les suivants :

- les composants classiques de l'API graphique ([Form](#), [Button](#), [Label](#), [Dialog](#), [MultiButton](#), etc.) ;
- [Tabs](#) ;
- [Toolbar](#) ;
- [ShareButton](#) ;
- [Preferences](#) ;
- [menu hamburger](#) ;
- [Pull to refresh](#) ;
- [réseau et services web](#).

1. Brève analyse de l'application à concevoir

L'application que nous allons créer se nommera *CN1FilmsBox* et permettra d'avoir des informations sur les derniers films populaires du moment y compris la liste des meilleurs films d'une année quelconque que l'utilisateur devra entrer au clavier. Le rendu de l'application finale sera semblable à celle des [Figure 10.1](#), [Figure 10.2](#) et [Figure 10.3](#).

Figure 10.1 : Interface d'accueil de *CN1FilmsBox*

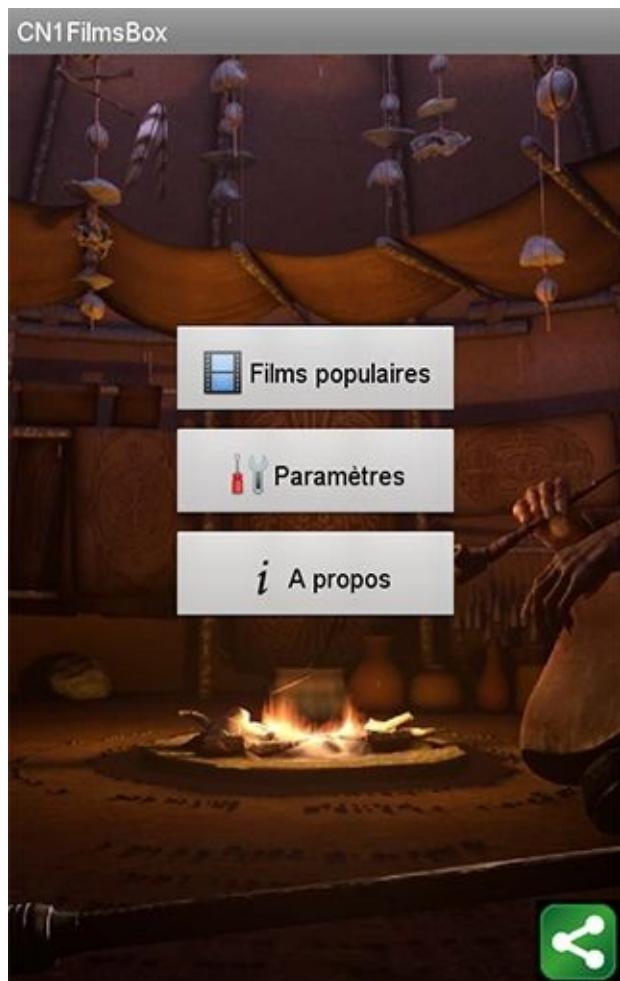


Figure 10.2 : Interface des films populaires de *CN1FilmsBox*

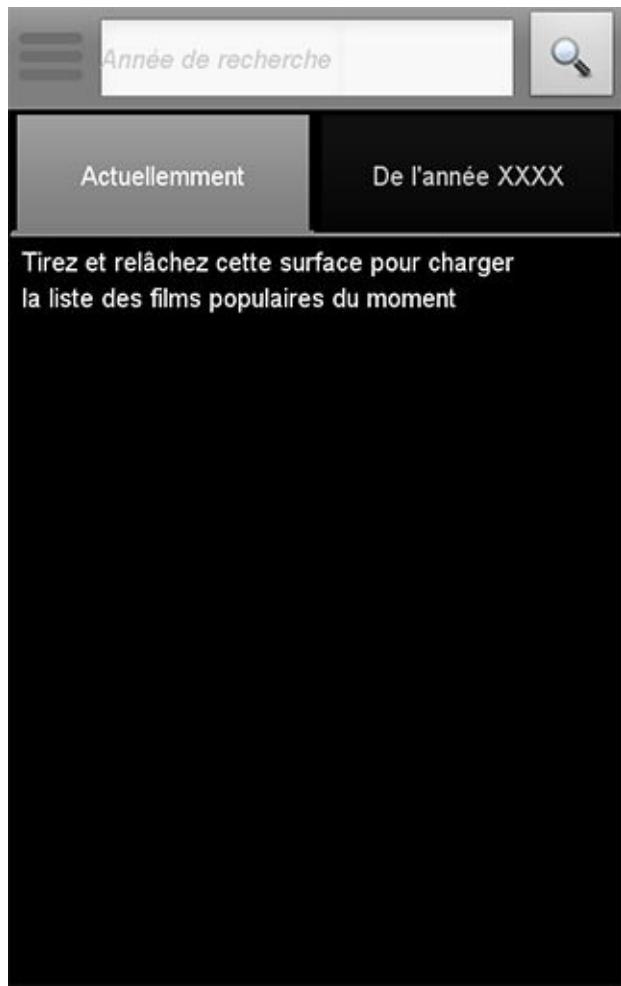
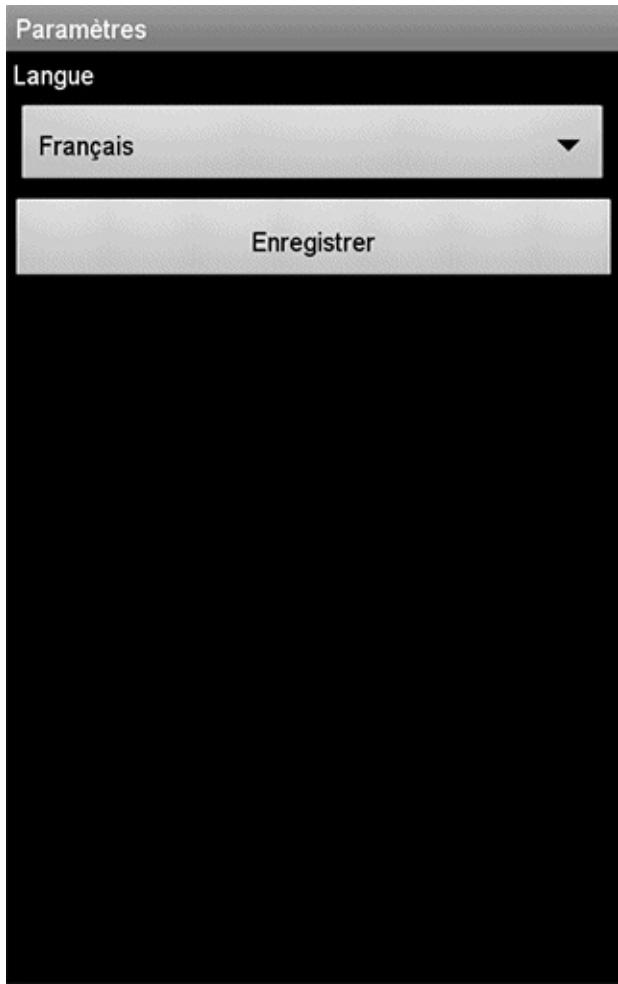


Figure 10.3 : Interface de paramétrage de la langue de CN1FilmsBox



Pour avoir accès à ces informations sur les films, nous allons faire appel au service web du site internet [the movie database](#). Ce site est une gigantesque base de données d'informations sur les films.

Pour accéder à son service web, vous devez d'abord [y ouvrir un compte](#) et récupérer une clé, qui se présente sous la forme d'un code alphanumérique. Toutes les requêtes que nous allons envoyer vers ce site utiliseront cette clé. Pour plus d'informations sur l'API, reportez-vous à sa [page de présentation](#) et sa [documentation](#).

Nous allons extraire sur chaque film les informations suivantes :

- son titre ;
- sa date de sortie ;
- sa note ;
- le nombre total de votes.

Toutes les images et icônes utilisées dans l'application sont ajoutées au fichier de ressources theme.res et le projet sera constitué de cinq classes qui contiendront la totalité du code. Le code a été rendu modulaire pour simplifier les futures modifications.

1.1. Les classes du projet

Voici ci-dessous le nom et la description globale du contenu de chacune des cinq classes du projet. Les descriptions seront détaillées dans les deux prochaines sections.

- `CN1FilmsBox.java` – Il s’agit de la classe principale du projet qui contiendra la structure de base de l’application, la création des instances des classes contenant les interfaces graphiques et les méthodes pour les afficher.
- `Accueil.java` – Cette classe contiendra l’interface d’accueil qui s’affichera en premier au lancement de l’application.
- `FilmsPopulaires.java` – Il s’agit de l’interface de recherche et d’affichage des listes des films qui seront chargées et affichées.
- `Parametres.java` – Cette classe contiendra une option qui permettra à l’utilisateur de changer la langue de l’application (français ou anglais).
- `RequeteReseau.java` – Cette classe contiendra la requête réseau que nous allons envoyer au service web. Elle héritera de la classe `ConnectionRequest` et contiendra aussi le code de récupération des résultats de la requête et le traitement des informations récupérées.

2. Cr ation de la classe principale et des interfaces graphiques

Cr ez un nouveau projet (avec le th me natif) qui contiendra par d faut la classe principale du projet (nomm e `CN1FilmsBox.java` dans notre cas). Nous reviendrons bient t sur ce fichier pour y ajouter du code. Avant cela, nous allons passer 脿 la cr ation des interfaces graphiques de l'application. La premi re sera celle de la premi re fen tre qui s'affichera quand l'application sera lanc e. Pour cela, ajoutez une classe nomm e `Accueil.java` au projet. Nous allons faire h riter cette derni re de la classe `Form`. La m me chose sera appliqu e  aux autres classes qui vont cr er les autres interfaces de l'application.

2.1. Interface principale (`Accueil.java`)

L'interface principale de la classe `Accueil` sera constitu e d'une image de fond, de trois boutons ordinaires, d'un bouton de partage et d'un menu Quitter pour fermer l'application. Ces trois boutons constituent le menu principal et permettent d'acc der aux fonctionnalit s de l'application.

Le premier nomm  `Films populaires` donne acc s 脿 une autre interface contenant une zone de recherche et deux onglets charg s de dresser la liste des films demand s. Le deuxi me bouton nomm  `Param tres` donne acc s 脿 une interface pour d finir la langue de l'application (le fran ais ou l'anglais). Le troisi me bouton nomm  `A propos` affichera une bo te de dialogue contenant des informations sur l'application (auteur et site du service web utilis ). Le quatri me bouton, qui est un bouton de partage cr   avec `ShareButton`, permettra 脿 l'utilisateur d'envoyer un message pr d fini (mais modifiable) 脿 ses proches pour leur faire d couvrir l'application.

Cette fen tre d'accueil aura un `BorderLayout` dans lequel nous allons ins rer deux conteneurs. L'un contiendra les trois boutons et sera ins r  au centre du `BorderLayout` et l'autre contiendra le bouton de partage qui sera ajout  en bas (au sud) du `BorderLayout`.

Chacune des images de cette interface (l'image de fond et l'image des ic nes des trois boutons) a 芅t  ajout e au fichier de ressources en tant que multi-images pour permettre la cr ation automatique d'autres r olutions de l'image pour le support de diverses r olutions d' cran.

Le code de cr ation de l'interface plac  dans le constructeur de la classe `Accueil` est le suivant :

```
public class Accueil extends Form implements ActionListener {  
  
    private CN1FilmsBox main;  
    private Command quitter;  
    private Button filmsPopul;  
    private Button aPropos;
```

```

private Button parametres;

public Accueil(CN1FilmsBox main){
    this.main=main;
    //Création d'un BorderLayout suivi de son ajout à la fenêtre
    BorderLayout mainLayout=new BorderLayout();
    mainLayout.setCenterBehavior(BorderLayout.CENTER_BEHAVIOR_CENTER);
    setLayout(mainLayout);
    setTitle("CN1FilmsBox");
    //---Ajout d'une image de fond à la fenêtre créée par Form
    getStyle().setBgImage(CN1FilmsBox.theme.getImage("Sintel2.jpg"));

    //Création d'un Container et de 3 boutons.
    Container actionsC=new Container(new BoxLayout(BoxLayout.Y_AXIS));
    filmsPopul=new Button("Films populaires",CN1FilmsBox.theme
    .getImage("Filmroll.png"));
    parametres=new Button("Paramètres",CN1FilmsBox.theme
    .getImage("parametres.png"));
    aPropos=new Button("A propos",CN1FilmsBox.theme.getImage("186.png"));

    //Ajout des 3 boutons au Container
    actionsC.addComponent(filmsPopul);
    actionsC.addComponent(parametres);
    actionsC.addComponent(aPropos);

    //Création d'un Container pour accueillir le bouton de partage
    Container partageC=new Container(new FlowLayout(Component.RIGHT));
    /* Création du bouton de partage et ajout au container
    précédemment créé */
    ShareButton partager=new ShareButton();
    if(CN1FilmsBox.getLangue().equals("fr")){
        partager.setTextToShare("Salut! CN1FilmsBox permet d'obtenir des
        informations sur les films les plus populaires du moment. "
        + "Elle délivre aussi la liste des meilleurs films d'une année
        quelconque.");
    } else {
        partager.setTextToShare("Hi! You can grab informations about
        popular films with CN1FilmsBox. You can also have the list of the best
        films of any year");
    }

    partageC.addComponent(partager);
    //Création d'une Command Quitter pour fermer l'application
    quitter=new Command("Quitter");
    addCommand(quitter);
    setBackCommand(quitter);

    addComponent(BorderLayout.CENTER,actionsC);
    addComponent(BorderLayout.SOUTH,partageC);

    /* Abonnement des 3 boutons et du Form au listener
    pour la réception d'événements */
    filmsPopul.addActionListener(this);
    parametres.addActionListener(this);
    aPropos.addActionListener(this);
    addCommandListener(this);
}

```

```

    }

public void actionPerformed(ActionEvent evt) {
    //....
}

}

```

2.2. Interface de recherche et d'affichage des films (FilmsPopulaires.java)

Cette interface est celle qui s'affichera quand l'utilisateur touchera le bouton Films populaires sur l'interface d'accueil. Elle contient deux onglets, une zone de texte et un bouton. Ces deux derniers serviront à lancer une recherche et sont situés dans la zone de titre de l'interface d'où l'utilisation du Toolbar. L'interface contient aussi un menu Retour pour retourner à l'interface d'accueil.

Le code de l'interface est le suivant :

```

public class FilmsPopulaires extends Form implements ActionListener {

    private static CN1FilmsBox main;
    private TextField zoneRecherche;
    private Button boutonRecherche;
    private Command retour;
    private Tabs tabs;
    private Container filmsPopuActuContainer;
    private Container filmsPopuAnContainer;

    public FilmsPopulaires(CN1FilmsBox main) {
        this.main=main;
        BorderLayout mainLayout=new BorderLayout();
        setLayout(mainLayout);

        /* Ces 2 méthodes créent un Container et y ajoutent du contenu.
        Les containers de ces méthodes seront ajoutés aux onglets
        qui seront créés plus bas */
        ongletFilmsPopulairesActu();
        ongletFilmsPopulairesAnnee();

        /* Création d'une page à onglets et ajout des 2 containers
        créés précédemment à l'intérieur des onglets */
        tabs=new Tabs();
        tabs.setSwipeActivated(false);
        tabs.addTab("Actuellement", filmsPopuActuContainer);
        tabs.addTab("De l'année XXXX", filmsPopuAnContainer);

        /* Création d'un contenu qui sera ajouté au toolbar
        créé plus bas. Ce contenu est composé d'un container,
        d'une zone de texte et d'un bouton */
        Container toolbarC=new Container(new BorderLayout());
        zoneRecherche=new TextField();

```

```

zoneRecherche.setHint("Année de recherche");
boutonRecherche=new Button(CN1FilmsBox.theme.getImage("rech.png"));
toolbarC.addComponent(BorderLayout.CENTER,zoneRecherche);
toolbarC.addComponent(BorderLayout.EAST,boutonRecherche);

/* Création de la commande Retour qui servira à retourner
à l'interface d'accueil */
retour=new Command("Retour");
setBackCommand(retour);

/* Création d'un Toolbar à placer sur la barre de titre.
Ce toolbar contiendra un Container qui aura une zone de texte
et un bouton.
Ajout de la commande Retour au toolbar qui se chargera
de l'ajouter à un menu hamburger */
Toolbar toolbar=new Toolbar();
setToolBar(toolbar);
toolbar.setTitleComponent(toolbarC);
toolbar.addCommandToSideMenu(retour);

addComponent(BorderLayout.CENTER,tabs);

boutonRecherche.addActionListener(this);
addCommandListener(this);
}

//....
}

```

2.3. Interface de paramétrage de la langue (Parametres.java)

L'interface de paramétrage est la plus simple des trois interfaces et est composée d'une liste déroulante et d'un bouton. Elle est affichée quand on clique sur le bouton Paramètres. La liste déroulante contient les deux langues supportées par l'application (le français et l'anglais). Le bouton sert à sauvegarder le choix effectué par l'utilisateur.

```

public class Parametres extends Form implements ActionListener{

private CN1FilmsBox main;
private ComboBox langues;
private Command retour;
private Button enregParam;

public Parametres(CN1FilmsBox main)
{
    this.main=main;
    setTitle("Paramètres");
    BoxLayout mainLayout=new BoxLayout(BoxLayout.Y_AXIS);
    setLayout(mainLayout);

    Label langLabel=new Label("Langue");

```

```
langues=new ComboBox();
langues.addItem("Français");
langues.addItem("Anglais");
enregParam=new Button("Enregistrer");

addComponent(langLabel);
addComponent(langues);
addComponent(enregParam);

retour=new Command("Retour");
setBackCommand(retour);

addCommand(retour);

enregParam.addActionListener(this);
addCommandListener(this);
}

//....
```


3. Implémentation des fonctionnalités

Les fenêtres étant maintenant créées, nous allons passer à l'ajout des fonctionnalités. Comme remarqué dans les codes précédents, les trois classes qui représentent les trois fenêtres de l'application héritent de l'interface ActionListener pour pouvoir recevoir les événements qui seront émis.

Comme pour les interfaces graphiques, nous allons maintenant passer à l'implémentation des fonctionnalités de chaque classe.

3.1. Interface principale (Accueil.java)

Au niveau de la fenêtre principale, nous traiterons les événements de clic des trois boutons et de la commande Quitter. Ces traitements se feront dans la méthode actionPerformed() héritée de l'interface ActionListener. Voici le contenu commenté de cette méthode.

```
public void actionPerformed(ActionEvent evt) {
    //Récupération de la commande qui a émis le signal
    Command cmd=evt.getCommand();
    //Récupération de l'objet qui a émis le signal
    Object obj=evt.getSource();

    if(obj==filmsPopul){
        /* Si le bouton appuyé est "Films populaires" alors afficher
        l'interface de la classe FilmsPopulaires */
        main.afficheFilmsPopulaires();

    } else if(obj==parametres){
        /* Si le bouton appuyé est "Paramètres" alors afficher
        l'interface de la classe Paramètres */
        main.afficheParametres();

    } else if(obj==aPropos){
        /* Affichage d'une boîte de dialogue avec le nom de l'auteur
        et le site de l'API web utilisée */
        Dialog.show("A propos", "informations_auteur",
Dialog.TYPE_INFO,CN1FilmsBox.theme.getImage("175.png"), "Ok", null);
    }

    if(cmd==quitter){
        //Si la commande Quitter est appuyée alors fermer l'application.
        Display.getInstance().exitApplication();
    }
}
```

3.2. Interface de recherche et d'affichage des films (FilmsPopulaires.java)

La méthode `actionPerformed()` de cette classe permettra de recevoir les événements du bouton de recherche et du bouton de retour. Le code commenté de cette méthode est le suivant :

```
public void actionPerformed(ActionEvent evt) {  
    Command cmd=evt.getCommand();  
    Object obj=evt.getSource();  
  
    if(obj==boutonRecherche){  
        /* Si le bouton de recherche est appuyé, appeler la méthode  
        chargerFilmsPopulairesAnnee() pour lancer la création  
        de la requête et la récupération des résultats */  
        chargerFilmsPopulairesAnnee();  
    }  
    /* Si la commande retour est appuyée, retourner à l'interface  
    principale */  
    if(cmd==retour){  
        main.afficheAccueil();  
    }  
}
```

En plus du constructeur et de la méthode `actionPerformed()`, la classe `FilmsPopulaires` contient cinq méthodes que voici avec leurs rôles.

- `void ongletFilmsPopulairesActu()` – Contient le code du contenu du premier onglet intitulé Actuellement. Elle est appelée dans le constructeur de la classe.
- `void ongletFilmsPopulairesAnnee()` – Contient le code du contenu du deuxième onglet intitulé De l'année XXXX. Elle est appelée dans le constructeur de la classe.
- `void chargerFilmsPopulairesActu()` – Contient le code qui crée et exécute la requête de recherche des films populaires du moment.
- `void chargerFilmsPopulairesAnnee()` – Contient le code qui crée et exécute la requête de recherche des meilleurs films de l'année entrée dans la zone de recherche placée dans la barre de titre.
- `static void afficheDetailsFilm(String titreFilm, String dateSortieFilm,
String noteFilm, String nbrevote)` – Il s'agit d'une méthode statique (qui sera appelée depuis la classe `RequeteReseau`) qui contient le code de création et d'affichage de l'interface qui affichera les détails sur le film choisi par l'utilisateur dans la liste des films établie.

Voici le code de chacune de ces méthodes :

```
private void ongletFilmsPopulairesActu()  
{  
    /* Création d'un container suivie de l'activation  
    du défilement vertical */  
    filmsPopuActuContainer=new Container(new BoxLayout(BoxLayout.Y_AXIS));  
    filmsPopuActuContainer.setScrollableY(true);  
  
    /* Création et affichage d'un message pour indiquer  
    à l'utilisateur l'action à exécuter */
```

```

SpanLabel info=new SpanLabel("Tirez et relâchez cette surface pour
charger la liste des films populaires du moment");
info.getStyle().setAlignment(Label.CENTER);
filmsPopuActuContainer.addComponent(info);

/* Si l'utilisateur tire et relâche la surface du Container alors
la méthode chargerFilmsPopulairesActu() sera appelée */
filmsPopuActuContainer.addPullToRefresh(new Runnable() {

    public void run() {
        chargerFilmsPopulairesActu();
    }
});
}

```

Dans cette méthode, nous faisons appel à `chargerFilmsPopulairesActu()` pour charger les films quand l'utilisateur va tirer (vers le bas) et relâcher la surface de l'onglet titré Actuellement.

```

private void ongletFilmsPopulairesAnnee()
{
    /* Création d'un container suivie de l'activation
    du défilement vertical */
    filmsPopuAnContainer=new Container(new BoxLayout(BoxLayout.Y_AXIS));
    filmsPopuAnContainer.setScrollableY(true);
}

private void chargerFilmsPopulairesActu(){
    /* S'il y a un composant dans le container du premier onglet
    alors les supprimer tous */
    if(filmsPopuActuContainer.getComponentCount()>0){
        filmsPopuActuContainer.removeAll();
    }
    /* Création de la requête HTTP à travers la classe RequeteReseau
    qui hérite de ConnectionRequest */
    RequeteReseau requete=new
    RequeteReseau(filmsPopuActuContainer,"https://api.themoviedb.org/3/movie/popular

    NetworkManager.getInstance().addToQueue(requete);

    InfiniteProgress progress=new InfiniteProgress();
    Dialog d=progress.showInifiniteBlocking();
    requete.setDisposeOnCompletion(d);
}

```

Dans cette méthode, nous créons une requête HTTP avec la classe `RequeteReseau` qui prend en premier paramètre le conteneur du premier onglet de l'interface graphique de la classe `FilmsPopulaires` et en deuxième paramètre l'URL à interroger. L'URL à interroger ici est `https://api.themoviedb.org/3/movie/popular` et l'URL complète qui sera envoyée après l'ajout de tous les paramètres aura la forme suivante :

`https://api.themoviedb.org/3/movie/popular?api_key=VOTRE_CLE_ICI.`

```

private void chargerFilmsPopulairesAnnee(){
    //Affichage du deuxième onglet
    tabs.setSelectedIndex(1);
}

```

```

String anneeRecherche=zoneRecherche.getText().trim();
if(anneeRecherche.length()==0 || anneeRecherche.length()!=4){
    Dialog.show("", "Entrez une année valide", "Ok", null);
    return;
}
/* S'il y a un composant dans le container du premier onglet
alors les supprimer tous */
if(filmsPopuAnContainer.getComponentCount()>0){
    filmsPopuAnContainer.removeAll();
}

/* Création de la requête HTTP à travers la classe RequeteReseau
qui hérite de ConnectionRequest.
Ajout de l'URL et de 3 paramètres et valeurs à la requête
sous forme d'arguments */
RequeteReseau requete=new RequeteReseau(
filmsPopuAnContainer, "https://api.themoviedb.org/3/discover/movie");
requete.addArgument("primary_release_year", anneeRecherche);
requete.addArgument("sort_by", "vote_average.desc");
requete.addArgument("language", "fr");
NetworkManager.getInstance().addToQueue(requete);

InfiniteProgress progress=new InfiniteProgress();
Dialog d=progress.showInifiniteBlocking();
requete.setDisposeOnCompletion(d);
}

```

Cette méthode est semblable à la précédente à cela près que c'est le conteneur du deuxième onglet qui est passé en paramètre au constructeur de RequeteReseau. Ici, l'URL à interroger est `https://api.themoviedb.org/3/discover/movie` et l'URL complète qui sera envoyée après l'ajout de tous les paramètres aura la forme suivante :

`https://api.themoviedb.org/3/discover/movie?`
`api_key=VOTRE_CLE_ICI&primary_release_year=ANNEE&sort_by=vote_average`
`.desc&language=fr .`

Le paramètre `primary_release_year` est l'année de recherche, `sort_by` permet de trier les résultats par ordre croissant ou décroissant. Ici, nous choisissons l'ordre décroissant avec la valeur `vote_average.desc`. Le dernier paramètre `language` qui prend la valeur `fr` permet de préciser que nous voulons des films disponibles en français. Le résultat affichera quand même des films en anglais si ces derniers n'ont pas été traduits en français, mais font quand même partie des meilleurs films de l'année recherchée. Pour plus de paramètres, consultez la [documentation de l'API de The Movie Database](#).

```

/* Cette méthode prend en paramètre les données retournées par la requête
HTTP, crée une nouvelle fenêtre et affiche ces données sur l'interface avec
des Label */
public static void afficheDetailsFilm(String titreFilm, String
dateSortieFilm, String noteFilm, String nbreVote)
{
    Form filmDetails=new Form(titreFilm);
    filmDetails.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
    SpanLabel titre=new SpanLabel("Titre: "+titreFilm);

```

```

Label date=new Label("Date de sortie: "+dateSortieFilm);
Label note=new Label("Note moyenne: "+noteFilm);
Label vote=new Label("Nombre de votes: "+nbreVote);

Command ret=new Command("Retour");
filmDetails.setBackCommand(ret);

filmDetails.addComponent(titre);
filmDetails.addComponent(date);
filmDetails.addComponent(note);
filmDetails.addComponent(vote);

filmDetails.addCommand(ret);
filmDetails.addCommandListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        main.afficheFilmsPopulaires();
    }
});

filmDetails.show();
}

```

3.3. Interface de paramétrage de la langue (Parametres.java)

Les clics sur le bouton Enregistrer et la commande Retour sont aussi gérés dans actionPerformed(). Le code est le suivant :

```

public void actionPerformed(ActionEvent evt) {
    /* Si le bouton Enregistrer est appuyé, sauvegarder l'indice
    de la langue choisie dans le Storage des préférences créées
    par la classe Preference */
    if(evt.getSource()==enregParam){
        Preferences.set("Lang_Pref", langues.getSelectedIndex());
        Dialog.show("Effectué!", "Changement effectué. Vous devez redémarrer
        l'application", "OK", null);
    }
    //Si le bouton Retour est appuyé alors retourner à l'interface
    if(evt.getCommand()==retour){
        main.afficheAccueil();
    }
}

```

Avant l'affichage de l'interface des paramètres, il faut réafficher sur celle-ci le dernier choix de langue effectué auparavant par l'utilisateur. Nous effectuons cela avec une autre méthode nommée chargerParametres().

```

public void chargerParametres(){
    //Chargement de l'indice de la langue choisie par l'utilisateur
    int choixLangue=Preferences.get("Lang_Pref", 0);
    /* Le choix de la langue par défaut est redéfini sur l'interface

```

```

    avec la valeur de l'indice chargée du Storage créé par la classe
    Preferences */
langues.setSelectedIndex(choixLangue);
}

```

3.4. Classe de la requête HTTP (RequeteReseau.java)

Pourquoi avoir créé une autre classe qui hérite de ConnectionRequest pour les requêtes au lieu de créer directement une requête de type ConnectionRequest ? C'est pour éviter d'avoir à répéter les mêmes codes de réception et de traitement des données envoyées par le service web pour les deux types de liste de films (films populaires du moment et meilleurs films d'une année) que nous voulons avoir. Ainsi, ces codes communs à la création de ces deux listes sont ajoutés dans cette classe. C'est aussi le cas de la clé de l'API que nous avons besoin d'utiliser pour les deux requêtes à créer.

Nous passons en paramètres au constructeur de cette classe un conteneur et une URL à interroger. À part la clé de l'API qui est ajoutée à l'intérieur du constructeur, les autres paramètres additionnels des requêtes seront ajoutés en dehors de cette classe.

Une fois les données sur les films récupérées, un MultiButton est créé pour chacun d'eux avec leur titre et leur date de sortie. Un clic sur l'un des MultiButton affichera une autre fenêtre avec la totalité des informations obtenues sur le film sélectionné.

Ci-dessous le code complet de cette classe.

```

public class RequeteReseau extends ConnectionRequest {

    private Container c;
    private int nbreResults=0;
    private List results;
    //Clé de l'API
    private static final String CLE_API="METTEZ VOTRE PROPRE CLE ICI";

    public RequeteReseau(Container c, String url)
    {
        this.c=c;
        //Ajout de l'URL à interroger
        setUrl(url);
        //Ajout de la méthode GET pour la récupération des données
        setPost(false);
        /* Ajout du paramètre api_key qui prend comme valeur
        la clé de l'API */
        addArgument("api_key", CLE_API);
    }

    @Override
    protected void postResponse() {
        nbreResults=results.size();

        for(int i=0;i<nbreResults;i++){

```

```

Map result=(Map)results.get(i);

final String titreFilm=(String)result.get("original_title");
final String dateSortieFilm=(String)result.get("release_date");
final double noteFilm=(Double)result.get("vote_average");
final double nbreVote=(Double)result.get("vote_count");

MultiButton infoFilm=new MultiButton();
infoFilm.setEmblem(CN1FilmsBox.theme.getImage("arrow.png"));
infoFilm.setTextLine1(titreFilm);
infoFilm.setTextLine2("Date de sortie: "+dateSortieFilm);
infoFilm.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        /* Appel de afficheDetailsFilm() qui va créer et
        afficher les informations qui lui sont passées
        en paramètres */
        FilmsPopulaires.afficheDetailsFilm(titreFilm,
dateSortieFilm, String.valueOf(noteFilm), String.valueOf(nbreVote));
    }
});
c.addComponent(infoFilm);
}
//Actualisation de l'interface
c.revalidate();
//Cette boîte de dialogue affiche le nombre de films téléchargés
Dialog.show("Terminé!",String.valueOf(nbreResults)+" films trouvés",
"Ok", null);
}

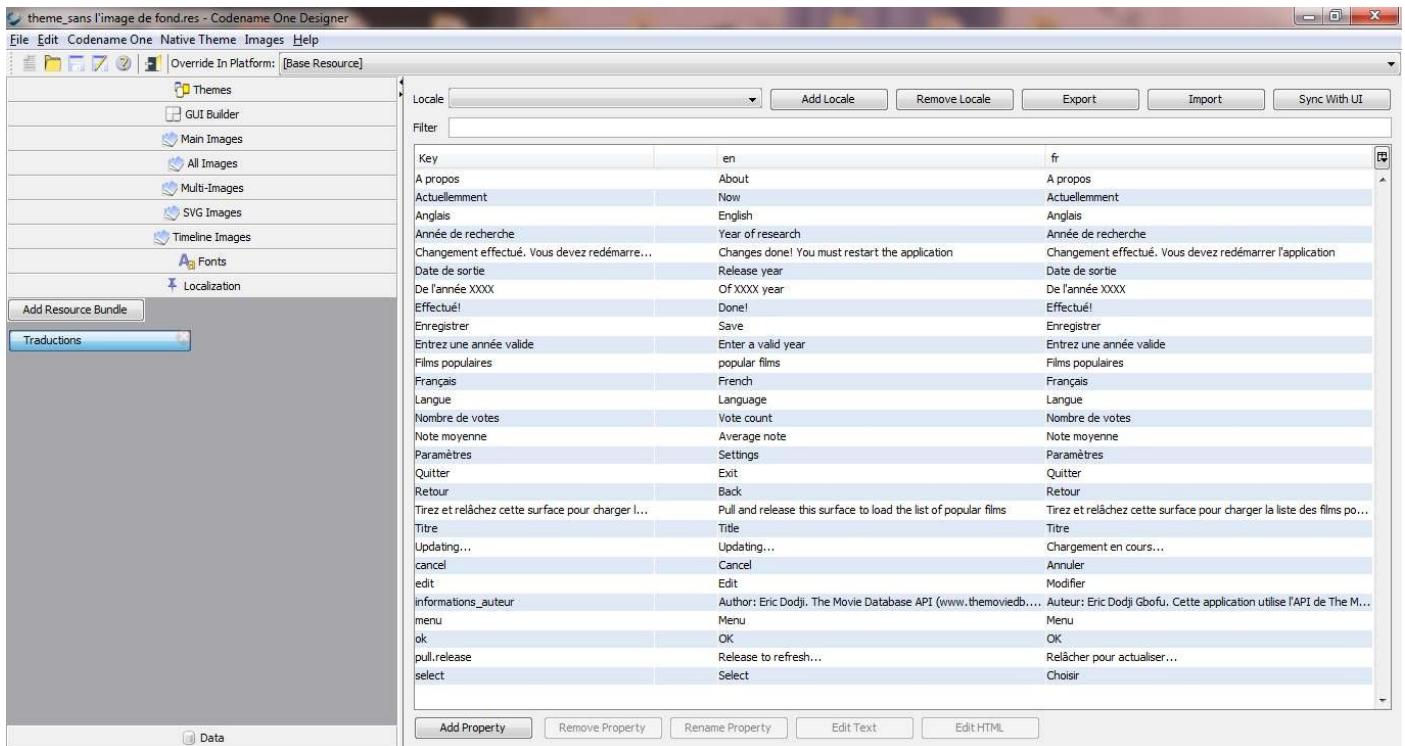
@Override
protected void readResponse(InputStream input) throws IOException {
JSONParser parser=new JSONParser();
Map data=parser.parseJSON(new InputStreamReader(input));
results=(List)data.get("results");
}
}

```

3.5. Traduction de l'application

Avant de passer au code de la classe principale du projet, nous allons créer un bundle de traduction pour y ajouter les textes de l'application en anglais et en français. Pour cela, ouvrez le fichier de ressources theme.res de votre projet puis sous l'onglet Localization, créez un nouveau bundle de traduction que vous allez nommer Traductions. Cliquez sur Add Locale pour ajouter une nouvelle langue (fr dans notre cas parce que celle de la langue anglaise intitulée en est toujours présente par défaut). Les détails sur la création d'un bundle de traduction et l'ajout des textes dans les diverses langues sont expliqués en détails à la [Section 7, Internationalisation/localisation](#). La [Figure 10.4](#) montre le contenu du bundle Traductions de notre application.

[Figure 10.4](#) : Contenu du bundle Traductions de CN1FilmsBox



Dans ce projet, la majorité des noms et textes à traduire sont les mêmes que ceux utilisés dans le code de l'application. Vous n'avez donc qu'à faire un copier-coller de ces textes dans les colonnes Key et fr. Pour finir, effectuez les traductions anglaises de ces phrases ou termes dans la colonne en et enregistrez les modifications.

3.6. Classe principale (CN1FilmsBox.java)

Cette classe est celle qui contient le code de la structure de base de notre application. Dans la méthode `init()`, nous chargeons la valeur de la langue sauvegardée par l'utilisateur et nous affichons la traduction correspondante à son choix. Si aucun choix n'a été trouvé alors la traduction française sera sélectionnée par défaut. Dans sa méthode `start()`, nous créons les instances des trois interfaces graphiques principales de l'application et nous affichons l'interface d'accueil. Nous créons aussi dans cette classe des méthodes publiques pour afficher ces interfaces graphiques. Voici son code:

```

public class CN1FilmsBox {

    private Form current;
    public static Resources theme;

    private Accueil accueil;
    private FilmsPopulaires fPopup;
    private Parametres parametres;
    private static String langue=null;

    public void init(Object context) {
        try {
            theme = Resources.openLayered("/theme");
            UIManager.getInstance().setThemeProps(

```

```

        theme.getTheme(theme.getThemeResourceNames()[0]));
    } catch(IOException e){
        e.printStackTrace();
    }

    //-----Code de localisation-----
    int choix_lang=Preferences.get("Lang_Pref", 0);
    if(choix_lang==0){
        langue="fr";
    } else if(choix_lang==1){
        langue="en";
    }
    Hashtable loc=theme.getL10N("Traductions", langue);
    UIManager.getInstance().setBundle(loc);
    //-----Code de localisation-----
}

public void start() {
    if(current != null){
        current.show();
        return;
    }

    accueil=new Accueil(this);
    fPopup=new FilmsPopulaires(this);
    parametres=new Parametres(this);

    accueil.show();
}

public void stop() {
    current = Display.getInstance().getCurrent();
}

public void destroy() {
}

public static String getLangue()
{
    return langue;
}

public void afficheAccueil(){
    accueil.show();
}

public void afficheFilmsPopulaires(){
    fPopup.show();
}

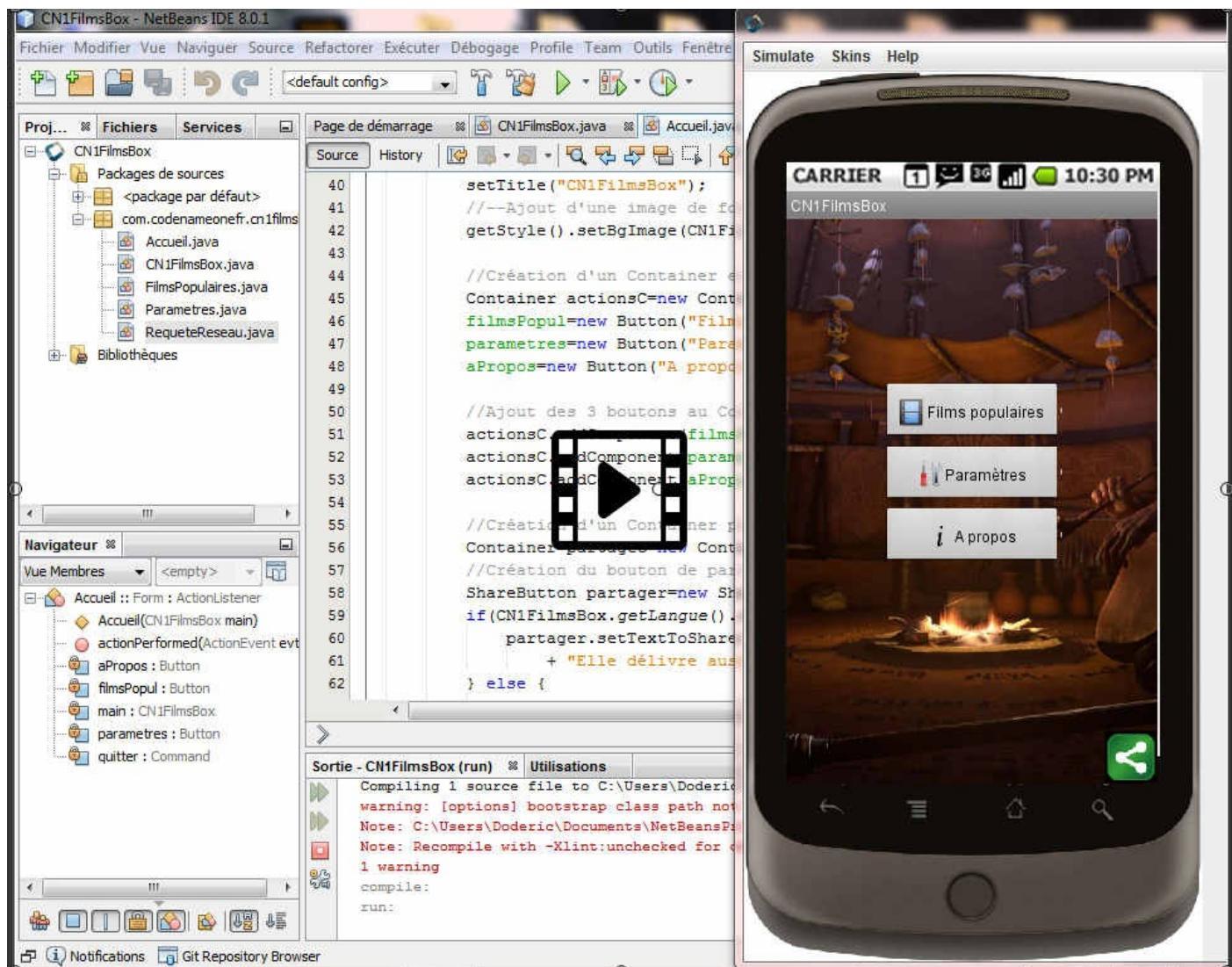
public void afficheParametres(){
    parametres.chargerParametres();
    parametres.show();
}
}
```


4. Idées d'extension de l'application

Comme exercice d'application, je vous demanderais d'ajouter l'affiche (ou le poster) des films aux informations extraites du service web. Vous pouvez aussi aller plus loin en proposant à l'utilisateur de voir la bande annonce d'un film qu'il a sélectionné si celle-ci est disponible. Vous n'avez qu'à regarder dans la [documentation de l'API de The Movie Database](#) pour repérer les arguments à ajouter à vos requêtes et les URL à utiliser.

Pour finir, la vidéo de la [Figure 10.5](#) montre notre application en action.

Figure 10.5 : CN1FilmsBox en action (vidéo)



5. Signature, compilation et déploiement

Notre application étant terminée, il va falloir maintenant passer à sa signature, à sa compilation et à son déploiement. L'[Annexe 3 : Signature d'une application](#) vous renseigne sur la procédure à suivre pour signer une application selon la plateforme ciblée. Et pour compiler votre projet, procédez comme expliqué à la [Section 4, La compilation avec Codename One](#).

Une fois la compilation effectuée et l'exécutable de votre application récupéré dans votre espace membre, vous pouvez l'installer sur votre appareil. Vous pouvez aussi le placer dans l'une des boutiques d'applications des plateformes que vous visez. Compte tenu de la quantité d'articles et de vidéos traitant de la publication d'applications sur les différentes boutiques des plateformes mobiles, nous n'avons pas abordé cet aspect dans cet ouvrage. Vous trouverez certainement votre bonheur en effectuant une petite recherche sur Internet.

Conclusion

Faire du développement mobile multiplateforme natif n'a jamais été aussi simple de nos jours comme c'est le cas avec Codename One. En trois ans seulement après sa sortie (de 2012 à 2015), on peut affirmer sans aucun doute que ce framework est destiné à un bel avenir. La rapidité de son évolution fascine et rassure. Il en est de même pour son business model qui est basé sur l'open source et le choix de l'utilisation du cloud. Créer une application mobile métier multiplateforme et native de surcroît avec Codename One est un régal, et il serait dommage de s'en priver si l'on est un développeur qui s'y connaît en langage Java.

Même si beaucoup de sujets ont été couverts dans ce livre, il en reste encore un nombre non négligeable qui ne l'ont pas été. C'est pourquoi et du fait de l'évolution continue de CN1, ce dernier sera périodiquement mis à jour pour vous permettre d'avoir accès aux dernières fonctionnalités du framework.

Vous n'avez pas fini de découvrir CN1. Pour cela, je vous invite à vous amuser avec cet outil le plus souvent possible en créant de plus en plus d'applications. Ce n'est que de cette manière que vous pourrez mieux comprendre son fonctionnement et sa particularité par rapport aux autres outils du même type dans le domaine du développement mobile multiplateforme.

1. Contribuer à la conception de Codename One

À l'exception des codes utilisés au niveau de la partie serveur (pour la compilation et pour les services proposés), le plug-in de Codename One est totalement open-source et sous licence GPL avec une [Classpath Exception](#). La Classpath Exception vous autorise à packager les binaires de CN1 avec le code de vos applications. Elle vous autorise à garder fermé le code source de vos applications tout en vous permettant aussi de les commercialiser. Si l'envie vous prend de contribuer à son évolution ou de créer des extensions, vous pouvez le faire et le soumettre sur le [compte GitHub de Codename One](#), qui héberge toutes les sources du projet.

2. Contribuer à la galerie d'applications

Exposer votre application dans la [galerie d'applications](#) a pour avantage de donner de la visibilité à votre application et à Codename One. Cela permet aussi à d'autres développeurs de se faire une idée de la qualité des applications développées avec ce framework.

3. Ressources complémentaires

La plupart des ressources de Codename One sont en anglais donc si vous n'avez aucun souci avec la langue de Shakespeare, foncez et apprenez tout ce qu'il y a à savoir sur le framework.

- Le [site officiel de Codename One](#) (en anglais) – Première source d'informations sur Codename One. Vous y trouverez un document PDF, le manuel officiel, des cours vidéo gratuits, des articles et un forum très actif.
- Le [site de la communauté francophone](#) – Première source d'informations en français sur Codename One. À l'instant où ces lignes sont écrites, vous trouverez sur ce site des articles et des cours vidéo. Vivement l'arrivée d'un forum en français !
- Le [blog de Steve Hannah](#) (en anglais) – Ce site n'est pas consacré exclusivement à Codename One, mais vous y trouverez régulièrement des articles sur le framework. Steve Hannah est l'un des développeurs de l'équipe de CN1. Avant de la rejoindre, il contribuait déjà beaucoup en plug-ins et en articles. Vous pouvez retrouver ses contributions personnelles sur son compte [GitHub](#).
- [Build Mobile iOS Apps In Java Using Codename One](#) – Cours vidéo gratuit créé par les concepteurs de Codename One (en anglais). Il s'agit principalement d'une étude de cas qui aide à concevoir une application mobile de partage d'images.
- [Learn Mobile Programming By Example With Codename One](#) – Cours vidéo gratuit créé par les concepteurs de Codename One (en anglais). Présentation d'une série d'applications dont les codes sont commentés.
- [Codename One 101](#) – Autre cours vidéo en anglais créé aussi par les concepteurs de Codename One. Ce cours couvre divers sujets et est conseillé si vous débutez à partir de zéro. Il est payant, mais reste gratuit pour les utilisateurs du compte PRO et supérieur.
- [Initiation à Codename One](#) – Cours vidéo en français. Il s'agit d'un cours d'initiation couplé à une étude de cas. Cette dernière concerne la création d'une application de gestion de tâches quotidiennes.
- La [Javadoc](#) – Documentation technique sur chaque classe de l'API.
- La [FAQ](#) – Il ne s'agit pas d'une page contenant des cours, mais de la foire aux questions. Vous y trouverez les questions et les réponses fréquemment posées sur Codename One.

Annexe 1 : Event Dispatch Thread (EDT)

1. Présentation de l'EDT

Codename One permet la création et l'utilisation de plusieurs processus légers (threads) à l'intérieur d'une application, mais pour bien interagir avec les composants graphiques, il faut que toutes les actions se passent à l'intérieur de son thread principal nommé EDT (Event Dispatch Thread). L'EDT représente le cœur de son architecture graphique et chaque action (événements, dessins d'interface, gestion du cycle d'exécution de l'application, etc.) se passera à l'intérieur de celui-ci. Le fait d'essayer de modifier les éléments de cette API graphique à l'extérieur de ce thread engendrerait des comportements bizarres et non définis.

Les seules actions qui peuvent se passer en dehors du processus EDT sont celles qui se passent à l'intérieur d'un thread créé par le développeur ou par les classes permettant d'interagir avec le réseau. Même si par défaut, la majorité des opérations se passent à l'intérieur de l'EDT, il est possible de le quitter et d'y revenir à volonté comme nous le verrons dans les deux prochaines sections.

La boucle d'exécution de l'EDT peut être représentée par ce pseudo code :

```
Tant que(Codename One est en exécution)
{
    gestionDesEvènements();
    dessinsDesComposantsEtAnimations();
    attendreLeProchainCycleDeLEDT();
}
```

Il existe deux causes majeures de violation de l'EDT :

- appel d'une méthode de l'API graphique de Codename One dans un autre thread (créé par les classes réseau ou créé par le développeur) différent de celui de l'EDT ;
- exécution d'une tâche complexe et coûteuse en CPU rendant lente l'exécution de l'application ou la bloquant tout simplement.

Pour prévenir certains comportements désagréables, il serait prudent en cas de doute de vérifier si l'on est toujours à l'intérieur de l'EDT ou non. Cette vérification s'effectue à l'aide de la méthode statique `Display.getInstance().isEDT()` qui retourne un booléen.

2. Exécution d'une action dans l'EDT depuis un autre thread (CallSerially/CallSeriallyAndWait)

Si la majorité des actions doivent se passer à l'intérieur de l'EDT et que le fait de faire appel à un élément de l'API graphique hors de ce thread bloque l'application ou engendre des comportements indéfinis, alors il va falloir trouver un moyen pour effectuer correctement ce genre d'action. Voici un cas précis. Vous effectuez un long téléchargement ou un long traitement de données et à la fin de cette action, vous devez notifier l'utilisateur en lui affichant un message de confirmation sur l'interface courante de l'application ou à travers une boîte de dialogue. Si nous affichons ce message ou cette boîte de dialogue depuis le code du thread alors nous allons violer l'EDT et nous pourrons remarquer un comportement indésirable de la part de l'application. Pour résoudre ce problème, nous allons faire appel à la méthode `callSerially()` du singleton Display. Cette méthode prend un Runnable en paramètre à l'intérieur duquel nous pouvons écrire le code d'affichage du message de confirmation de la fin du téléchargement ou du traitement. Voici un exemple de code qui illustre notre cas :

```
/* Codes et actions s'exécutant dans un autre thread
   (Ex. Téléchargement ou traitement de données)
...
Display.getInstance().callSerially(new Runnable() {

    public void run() {
        // les actions écrites à cet endroit s'exécuteront à l'intérieur de
        // l'EDT
        Dialog.show("Confirmation", "Traitement ou téléchargement terminé",
        "OK", null) ;
    }
});
/* Codes et actions s'exécutant dans un autre thread
   (Ex. Téléchargement ou traitement de données)
...
}
```

Dans la même logique que `callSerially()`, `callSeriallyAndWait()` permet aussi d'exécuter du code dans l'EDT depuis un autre thread. La différence entre ces deux méthodes est que cette dernière bloque le thread courant jusqu'à la fin de l'opération qu'elle doit exécuter avant de retourner. Utilisez alors `callSeriallyAndWait()` si vous ne voulez pas qu'une action s'exécute en parallèle au code à exécuter dans celle-ci. Ayant le même prototype que `callSerially()`, elle s'utilise de la même manière que cette dernière.

3. Exécution d'une action dans un autre thread depuis l'EDT (InvokeAndBlock)

À l'inverse de `callSerially()`, la méthode `invokeAndBlock()` permet de créer un thread pour exécuter une action pendant qu'on est à l'intérieur de l'EDT. Dans ce cas, l'EDT sera temporairement bloqué pour laisser place à l'exécution de l'action écrite dans `invokeAndBlock()`.

```
//.... Actions s'effectuant dans le thread principal (EDT)
Display.getInstance().invokeAndBlock(new Runnable() {

    public void run() {
        //...Actions s'effectuant dans un autre thread
    }
}); //.... Actions s'effectuant dans le thread principal (EDT)
```

Encadré : Autres ressources (en anglais) sur l'EDT

Deux articles détaillés parlent de l'EDT sur le blog de Codename One : [CallSerially The EDT & InvokeAndBlock - part 1](#) et [CallSerially The EDT & InvokeAndBlock - part 2](#).

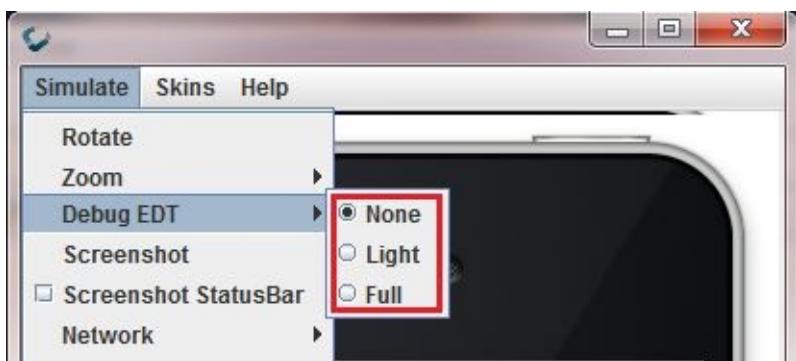
Vous pouvez aussi lire d'autres informations sur le sujet dans le manuel en ligne au chapitre [The EDT - Event Dispatch Thread](#).

Note > Une version PDF du [manuel officiel](#) est téléchargeable.

4. Détection des violations de l'EDT

Pour simplifier la détection des violations de l'EDT dans un programme, une fonctionnalité est fournie dans le simulateur de Codename One. Si cette fonctionnalité est activée, un message de violation de l'EDT s'affichera dans la console de l'environnement de développement chaque fois qu'un cas de violation se présente. Pour activer cette fonctionnalité, allez dans le menu Simulate puis Debug EDT du simulateur puis choisissez l'option Light ou Full (voir [Figure .176](#)). L'option None (activée par défaut) permet de désactiver cette fonction. L'option Light affichera seulement un avertissement en cas de violation et l'option Full affichera un rapport complet contenant des détails sur l'endroit où la violation s'est produite.

Figure .176 : Outil de débogage de l'EDT

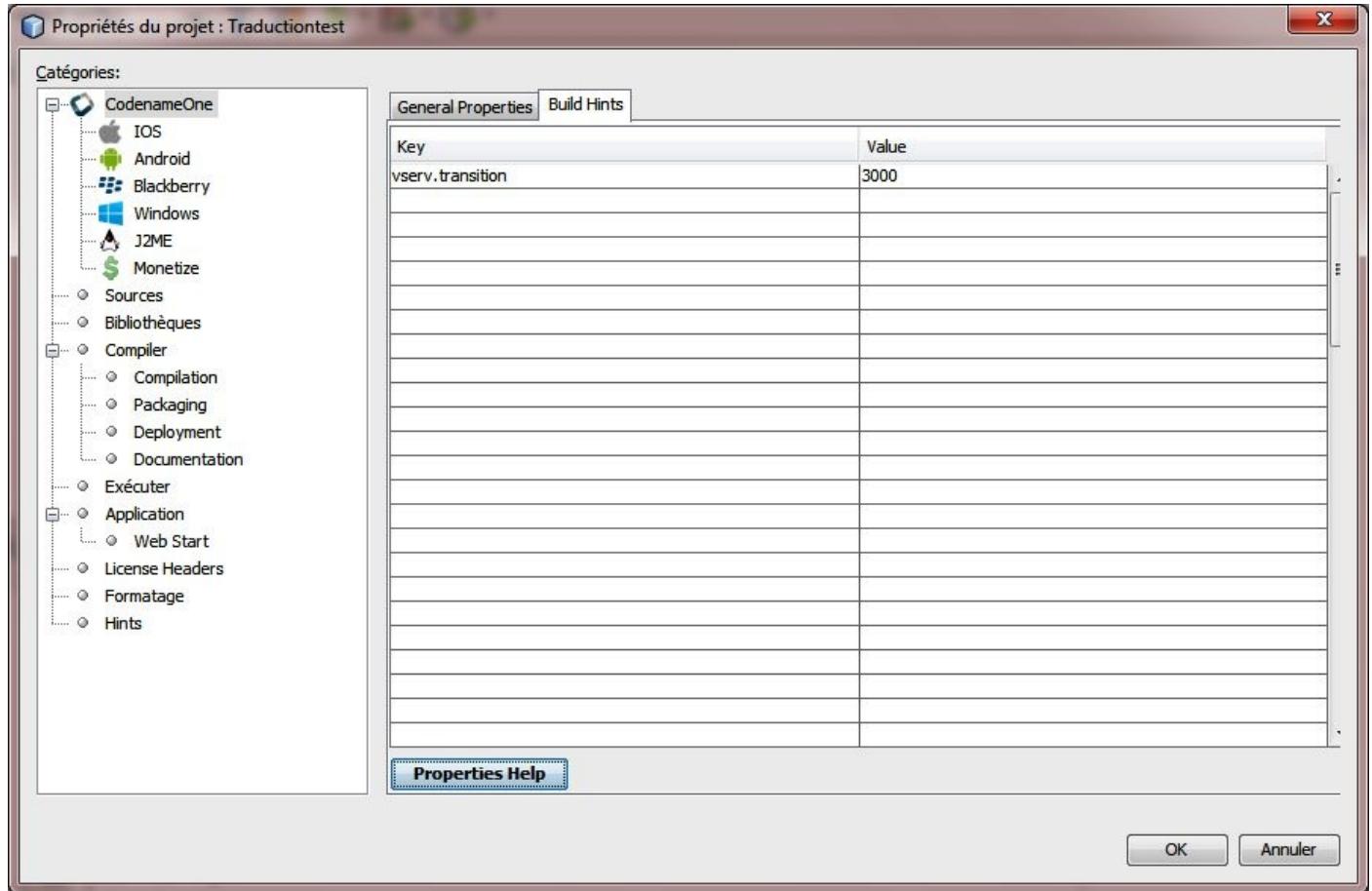


Attention > Cet outil ne donne pas toujours de vrais résultats et pourrait signaler de faux cas de violations de l'EDT mais il sera d'une grande aide parce que ce genre de problème peut se révéler complexe à traquer soi-même en temps normal.

Annexe 2 : Les arguments de compilation

À divers endroits de cet ouvrage, nous avons vu qu'en plus du code, il fallait ajouter parfois des clés et des valeurs dans les propriétés des projets sous l'onglet Build Hints (voir la [Figure .177](#)) avant de lancer la compilation. Ces clés sont appelées des *arguments de compilation*. Ce sont des paramètres additionnels qu'on peut ajouter au processus de compilation qui se passe dans le cloud.

Figure .177 : Interface d'ajout des arguments de compilation



La liste suivante des arguments est incomplète et ne contient que les arguments des besoins les plus courants. Pour consulter la liste complète et suivre sa mise à jour, faites un tour sur la page [Advanced Build](#) (en anglais) du site de Codename One.

1. Tableaux récapitulatifs des arguments de compilation

Tableau .8 : Quelques arguments de compilation de l'iOS

Argument	Valeurs possibles
Description	
ios.project_type	ios, ipad, iphone. Valeur par défaut : ios
Précise si l'application qui sera générée sera exécutée sur l'iPhone, l'iPad ou sur les deux.	
ios.application_exits	true ou false. Valeur par défaut : true
Précise si l'application doit se fermer si on appuie sur la seule touche physique présente sur les iPhone et les iPad.	
ios.interface_orientation	UIInterfaceOrientationPortrait, UIInterfaceOrientationPortraitUpsideDown, UIInterfaceOrientationLandscapeLeft, UIInterfaceOrientationLandscapeRight. Valeur par défaut : UIInterfaceOrientationPortrait
Indique à Codename One de verrouiller l'orientation (mode portrait ou paysage) de l'interface de l'application dans celle qui sera spécifiée. Par défaut, cette orientation est libre et s'adapte à l'orientation de l'écran.	
ios.statusbar_hidden	true ou false. Valeur par défaut : false
Masque ou affiche la barre de statut si la valeur est placée à true.	
ios.includePush	true ou false. Valeur par défaut : false
Précise l'utilisation des notifications push.	

Tableau .9 : Quelques arguments de compilation d'Android

Argument	Valeurs possibles
Description	
android.debug	true ou false. Valeur par défaut : true
Inclut ou exclut la création de la version debug de l'exécutable de l'application pendant la compilation.	
android.release	true ou false. Valeur par défaut : true
Inclut ou exclut la création de la version release de l'exécutable de l'application pendant	

la compilation.

android.min_sdk_version	Valeur par défaut : 7
Précise la version minimum du SDK d'Android à utiliser pour la compilation. Cette valeur est ajoutée à la propriété android:minSdkVersion du fichier manifest d'Android.	
android.xpermissions	N/A
Ajoute des permissions additionnelles au fichier manifest d'Android.	
android.statusbar_hidden	true ou false. Valeur par défaut : false
Masque ou affiche la barre de statut si la valeur est placée à true.	
android.web_loading_hidden	true ou false. Valeur par défaut : false
Active ou désactive l'affichage de l'indicateur de progression pendant le chargement d'une page web.	
android.theme	Light ou Dark. Valeur par défaut : Light
Précise l'utilisation du thème Holo Light (qui est clair) ou Holo Dark (qui est sombre) sur les versions 4 ou supérieure d'Android.	

Tableau .10 : Quelques arguments de compilation de BlackBerry OS

Argument	Valeurs possibles
Description	
rim.askPermissions	true ou false. Valeur par défaut : true
L'accès aux fonctionnalités natives de BlackBerry OS déclenche des demandes de permission à l'utilisateur. Cet argument permet de le désactiver si sa valeur est false. Des demandes de permission fréquentes peuvent agacer l'utilisateur donc je conseille de mettre cette valeur à false.	
rim.ignor_legacy	true ou false. Valeur par défaut : false
Si la valeur de cet argument est passée à true alors l'exécutable créé sera pour les versions 5.0 ou supérieure de BlackBerry OS. Si elle est à false, l'exécutable ciblera aussi les versions 4.X.	
rim.nativeBrowser	true ou false. Valeur par défaut : false
Active l'utilisation du navigateur natif sur BlackBerry OS version 5.0 ou supérieure. La valeur false par défaut permet d'utiliser le HTMLComponent au lieu du composant natif.	

Tableau .11 : Quelques arguments de compilation divers

Argument	Valeurs possibles
Description	
block_server_registration	true ou false. Valeur par défaut : false
	Par défaut, chaque application créée avec Codename One envoie certaines informations vers le serveur cloud. Ces informations sont récupérées pour des besoins statistiques. Si vous ne voulez pas que votre application contribue à ces statistiques alors, mettez la valeur de cet argument à true.
crash_protect	true ou false. Valeur par défaut : true
	Cet argument ne concerne que les utilisateurs payants. Il permet d'activer et de désactiver l'envoi des rapports de crash vers le serveur.
noExtraResources	true ou false. Valeur par défaut : false
	Pour certains besoins, le serveur de Codename One injecte ses propres ressources à l'exécutable des applications. Mettre la valeur de cet argument à true permet de désactiver cette action. Cela a pour effet de réduire la taille des exécutables.
j2me.iconSize	Valeur par défaut : 48x48
	Définit la taille de l'icône des applications sur les plateformes J2ME. Il s'agit de préciser la taille de la largeur × la taille de la hauteur.

Annexe 3 : Signature d'une application

Avant la compilation et le déploiement, une application mobile doit être numériquement signée. Cette signature se fait à l'aide d'un fichier nommé *certificat* et qui contient une clé privée. L'utilité de signer une application est de pouvoir identifier le concepteur de l'application en plus de créer une relation de confiance entre ce concepteur et le système d'exploitation (à travers son application). La création d'un certificat varie en fonction de chaque plateforme mobile. Elle est gratuite sous certaines plateformes et payante sous d'autres. De manière générale, toutes les applications d'un développeur ou d'une entreprise sont signées avec le même certificat.

Attention > *Une application non signée ne s'installera pas sur les appareils.*

1. Sous Android

La création d'un certificat pour la plateforme Android est gratuite et est la plus simple. Sous Codename One, il est possible de le faire à partir de la fenêtre de propriétés de votre projet. Ce certificat contiendra des informations sur le développeur (ou la société) qui a conçu l'application. Pour le créer, vous devez effectuer l'action suivante.

Allez dans les propriétés de votre projet (clic droit sur le projet puis entrée Propriétés) et cliquez sur la sous-catégorie Android sur votre gauche. Cliquez ensuite sur le bouton Generate pour ouvrir une fenêtre (voir [Figure .178](#)) contenant des champs à remplir. Remplissez ces champs et cliquez sur le bouton Generate de cette fenêtre. Votre certificat est maintenant créé et est disponible sous forme d'un fichier d'extension .ks à l'emplacement que vous avez choisi.

Pour finir, cliquez sur le bouton Browse du champ Keystore pour sélectionner le fichier .ks sur votre ordinateur. Entrez ensuite le mot de passe dans le champ Keystore password puis l'alias dans le champ Keystore alias. Ces informations sont celles que vous avez entrées précédemment pendant la création du certificat. Le champ Generate Builds permet de préciser si vous voulez générer la version debug, release (option cochée par défaut) ou les deux (Both) de l'application. Validez votre paramétrage avec le bouton OK (voir [Figure .179](#)).

Figure .178 : Signature d'une application Android (étape 1)

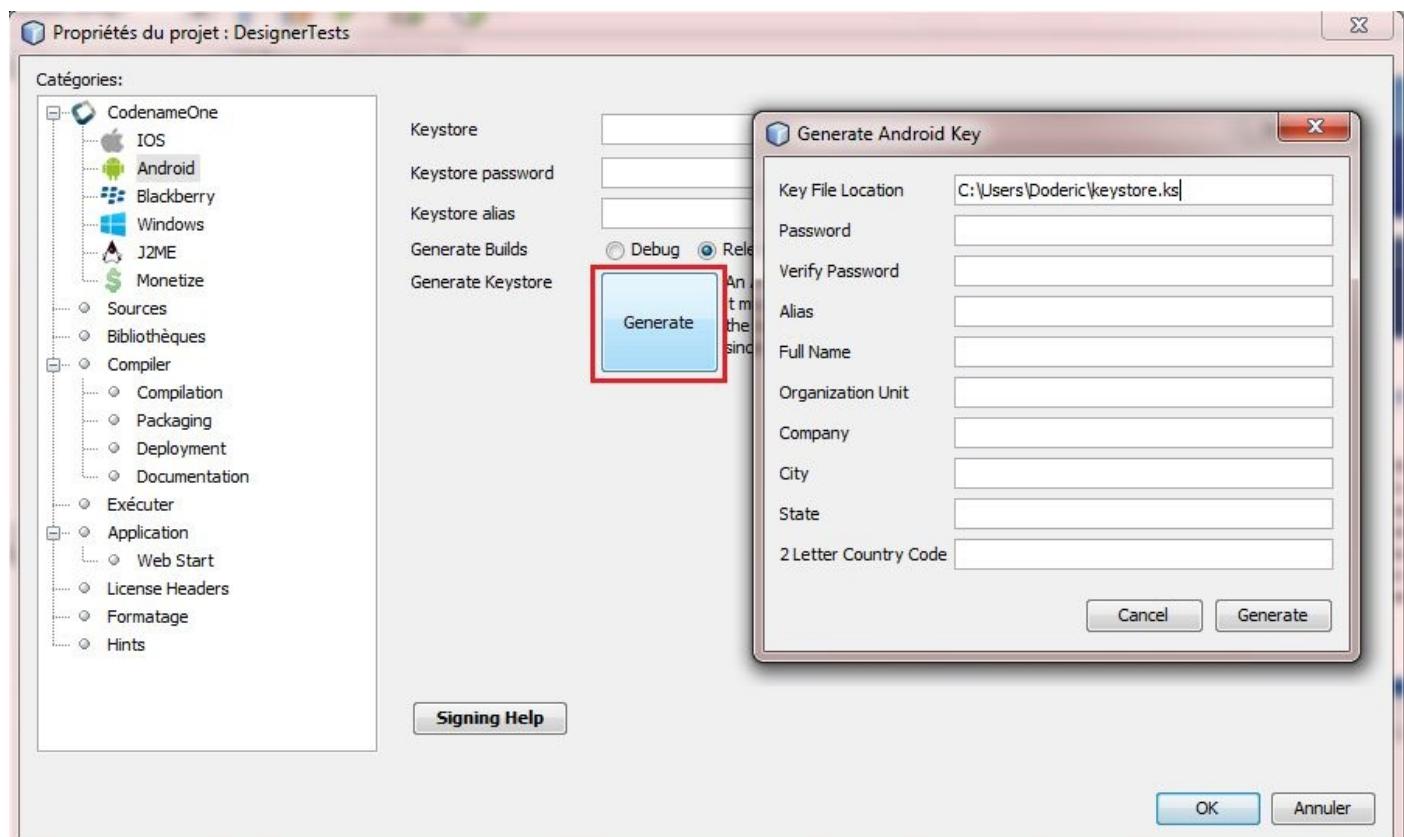
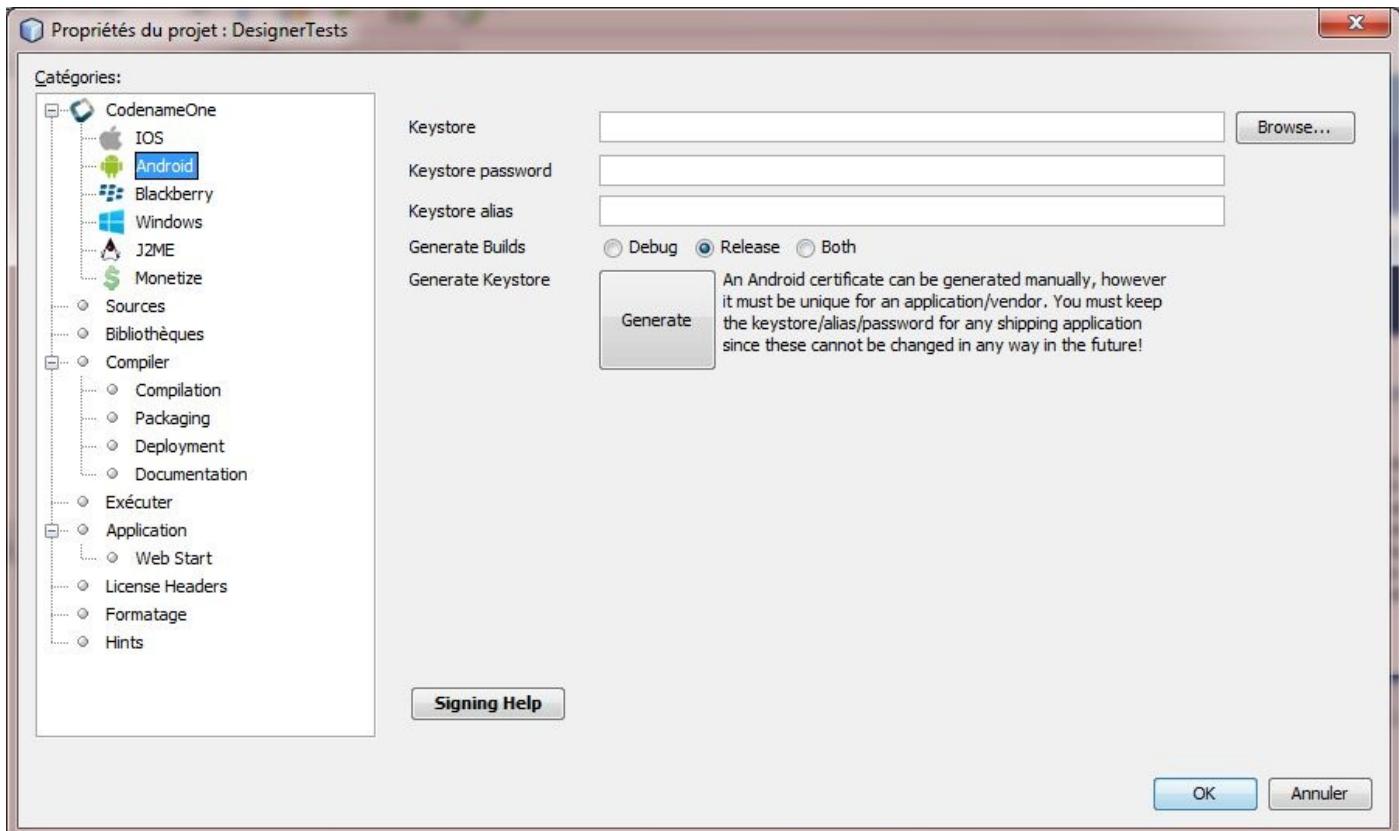


Figure .179 : Signature d'une application Android (étape 2)



Note > Pour plus d'informations sur la signature d'applications Android, consultez le [site officiel des développeurs Android](#).

2. Sous iOS

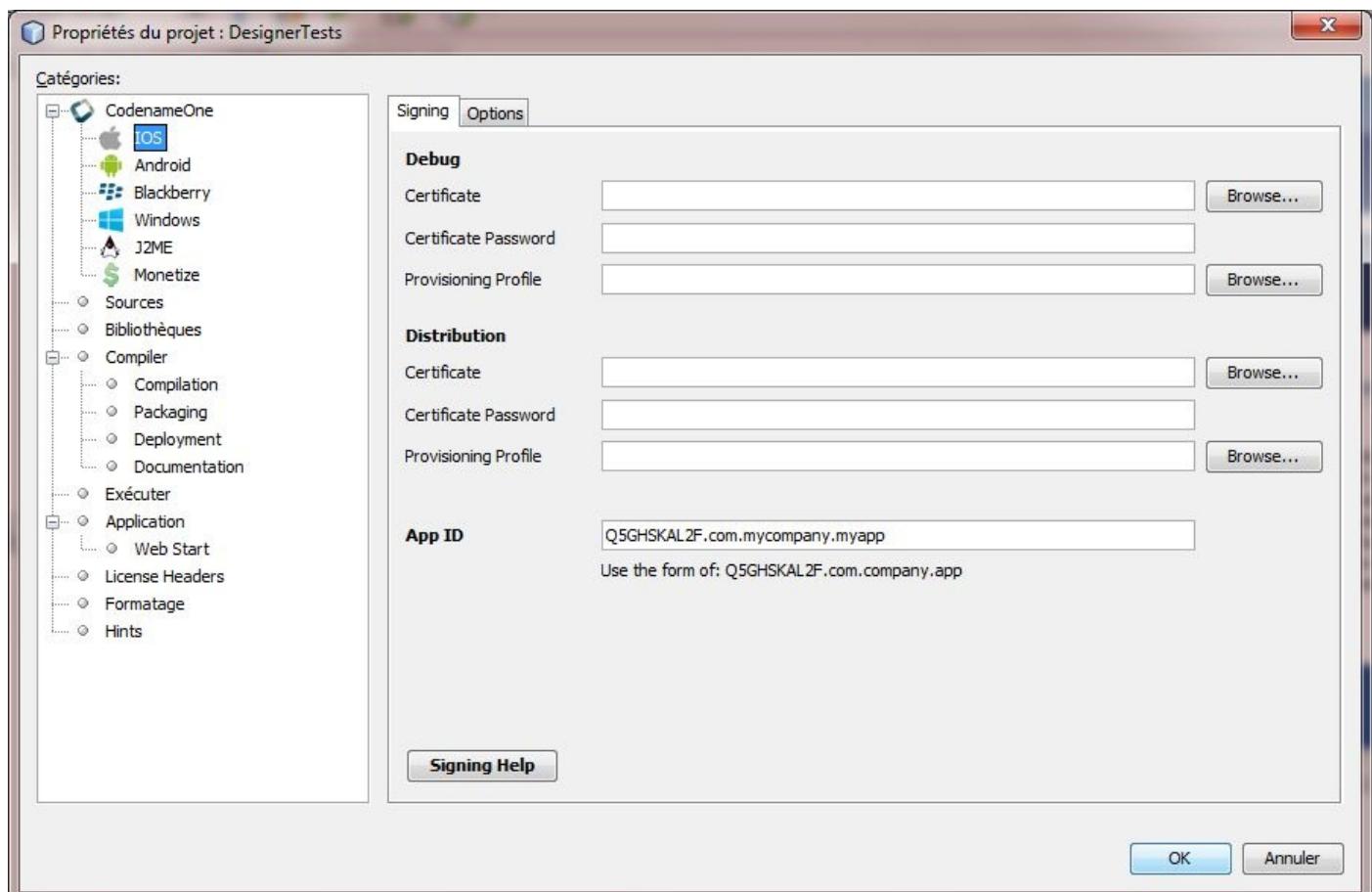
Avant toute chose, il est important de savoir que l'obtention d'un certificat pour la plateforme iOS est payante. Son prix s'élève à \$99 par an. Vous devez d'abord rejoindre le [programme des développeurs iOS](#) pour vous inscrire et vous acquitter de ce montant.

Pour la signature d'une application iOS, vous aurez besoin d'un certificat et d'un fichier nommé *provisioning profile*. Pour chacun de ces deux fichiers, vous aurez besoin d'une version de développement (pour pouvoir effectuer les tests de l'application sur votre appareil) et d'une version de distribution (pour le déploiement en ligne). Cela fait quatre fichiers au total.

Pour plus d'informations sur la création et la récupération de ces fichiers, reportez-vous à la [section dédiée du manuel en ligne de Codename One](#). Vous trouverez aussi des tutoriels vidéo détaillés sur le [site de Apple](#).

Une fois les fichiers de certificat et de provisioning profile récupérés, allez sous la sous-catégorie iOS de la fenêtre de propriétés de votre projet (voir [Figure .180](#)). Ajoutez les fichiers téléchargés dans les champs Certificate et Provisioning Profile et entrez votre mot de passe dans les champs Certificate Password.

Figure .180 : Signature d'une application iOS



Attention > La création du certificat et du provisioning profile pour l'iOS doit se faire obligatoirement sous un ordinateur Mac.

Astuce > Si vous n'avez pas d'ordinateur Mac, vous n'avez pas besoin d'en acheter ou d'en emprunter un pour créer vos certificats et envoyer votre application sur l'App Store. Le site [MacinCloud](#) permet à n'importe qui d'utiliser un Mac dans le cloud à partir d'un navigateur sous Windows ou à partir d'une application dédiée téléchargeable sur le site. Une offre gratuite de test est disponible avant de souscrire à l'un des packages payants.

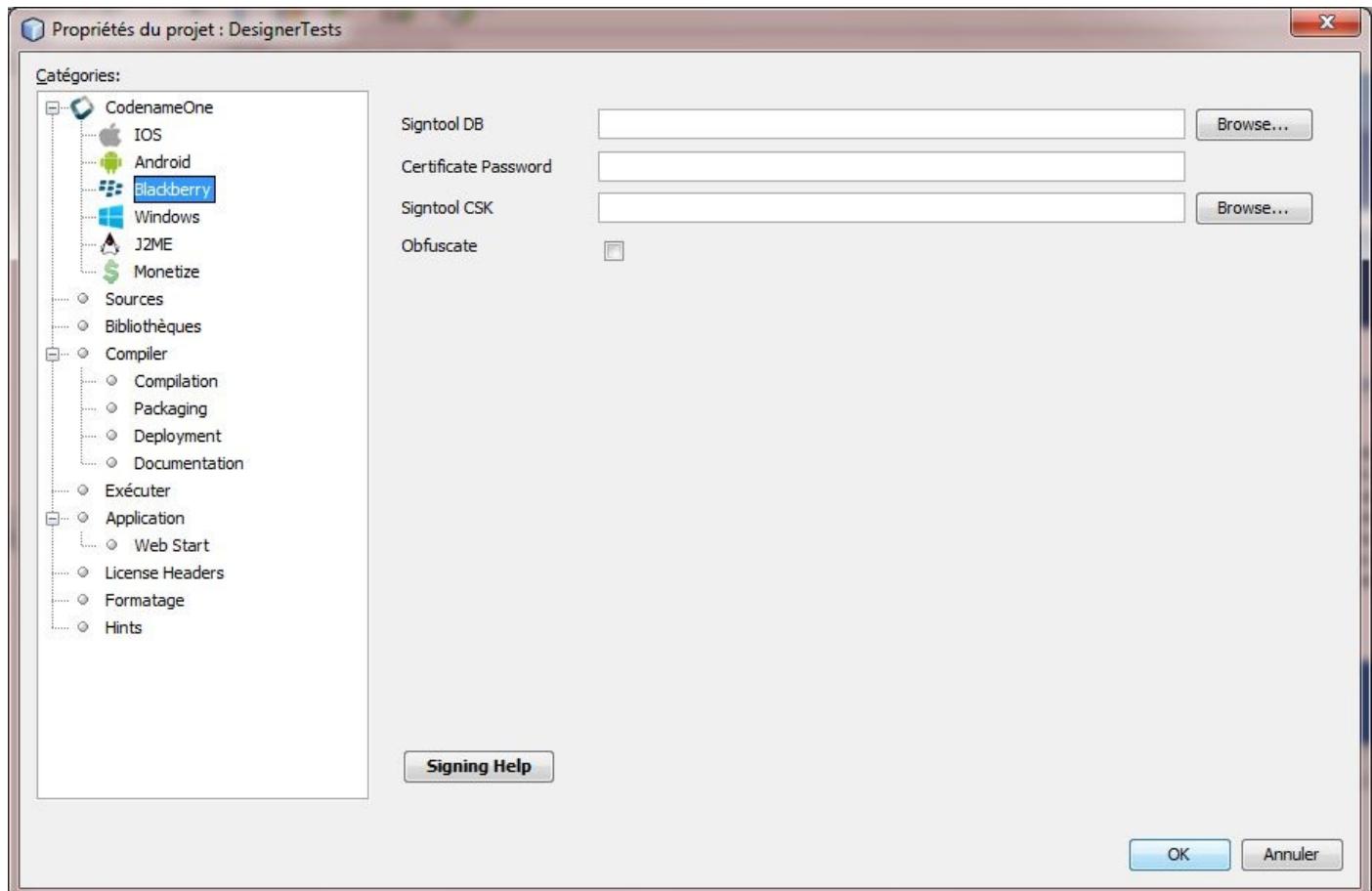
3. Sous Blackberry

Tout comme pour Android, la signature et l'accès à la boutique d'applications de Blackberry nommée Blackberry World est gratuite. Dans le cas de Blackberry, vous aurez besoin de deux fichiers nommés sigtool.db et sigtool.csk. Ces fichiers ne sont valables que pour les versions 4, 5, 6 et 7 de l'OS de Blackberry. La première chose à faire pour créer ces deux fichiers est d'aller sur la [page de demande de clés](#). Une fois que vous aurez rempli le formulaire disponible sur cette page, des instructions vous seront envoyées par mail pour la suite de la procédure. Dans ce processus de création des fichiers d'extension .db et .csk, vous aurez besoin du [BlackBerry JDE \(Blackberry Java Development Environment\)](#) qui est l'environnement de développement pour les plateformes Blackberry de la version 4 à 7.

Note > Pour plus d'informations sur la signature d'applications Blackberry, consultez la [page d'aide sur BlackBerry Jam Zone](#).

Les fichiers de signature étant obtenus, allez sous la sous-catégorie Blackberry de la fenêtre des propriétés du projet (voir la [Figure .181](#)). Ajoutez le fichier sigtool.db dans le champ Signtool DB et le fichier sigtool.csk dans le champ Signtool CSK. Ajoutez aussi le mot de passe du certificat dans le champ Certificate Password puis validez avec OK pour fermer la fenêtre.

Figure .181 : Signature d'une application Blackberry



4. Sous Windows Phone

La signature des applications Windows Phone n'est pas dans les attributions du développeur. Une fois son application validée, c'est le Windows Marketplace qui la signera avant de la rendre accessible dans la boutique. En résumé, le développeur n'aura pas à s'occuper lui-même de la création d'un certificat. Tout comme celui de l'App Store d'Apple, l'accès au Windows Marketplace est payant.

Index

A

ActionListener, Exemple : Page avec titre et menus, Button, CheckBox, Exemple : Créer des boutons d'enregistrement, Gestion de la connexion, Avec NetworkManager et ConnectionRequest, Avec ImageDownloadService, Quelques méthodes de NetworkManager, Utilisation d'un service web quelconque, Interface principale (Accueil.java)

AdMob, Google Mobile Ads

Adspot, Vserv

Alignement, Quelques méthodes de FlowLayout

texte dans label, Exemple : Afficher un texte court ou une image

texte dans Multibutton, Quelques méthodes de MultiButton

AnalyticsService, Rapports statistiques Animation, Transitions Appel, Appel, e-mail, SMS Application

paiement depuis, Effectuer un paiement depuis une application

publicité dans, Monétisation des applications gratuites

structure de base, Structure d'une application Codename One

thème, Hello Codename One

Arbre hiérarchique, Envoi de données (upload) avec MultipartRequest Argument de

compilation, Annexe 2 : Les arguments de compilation Arrière-plan, Couleurs et

images AutoCompleteTextField, AutoCompleteTextField

B

Barre

d'outils, Toolbar

de progression, Slider, Avec NetworkManager et ConnectionRequest

de titre, Toolbar

Base de données, Avec Database (pour les bases de données SQLite), Communication avec une base de données distante Boîte de dialogue, Dialog BorderLayout, BorderLayout, Interface principale (Accueil.java) Bouncy Castle, Des plug-ins gratuits à votre disposition Bouton, Button, Interface de recherche et d'affichage des films (FilmsPopulaires.java)

de partage de données, ShareButton

états, Quelques méthodes de Button

Marche/Arrêt, [OnOffSwitch](#)

BoxLayout, [BoxLayout](#)BrowserComponent, [WebBrowser](#)Button, [Button](#), Interface de recherche et d'affichage des films ([FilmsPopulaires.java](#))

méthodes, [Quelques méthodes de Button](#)

ButtonGroup, [RadioButton](#)

méthodes, [Quelques méthodes de ButtonGroup](#)

C

Calendar, [Calendar](#)

Calendrier, [Calendar](#), Des plug-ins gratuits à votre disposition

Capture, [Capture](#)

Capture d'écran, [Le menu Simulate](#)

Case à cocher

à choix multiple, [Quelques méthodes de MultiButton](#), [CheckBox](#)

à choix unique, [Quelques méthodes de MultiButton](#), [RadioButton](#)

Charger

un fichier local, [Exemple : Lire une page HTML locale ou distante](#)

un site en ligne, [Exemple : Lire une page HTML locale ou distante](#)

une image, [Exemple : Afficher un texte court ou une image](#)

CheckBox, [CheckBox](#)

méthodes, [Quelques méthodes de CheckBox](#)

Cloud

Codename One, [Pourquoi Codename One ?](#)

Compilation, [La compilation avec Codename One](#)

envoi d'e-mails, [Appel, e-mail, SMS](#)

envoi des journaux, [Logging](#)

stockage, [Avec CloudStorage \(pour le cloud\)](#)

CloudObject, [Avec CloudStorage \(pour le cloud\)](#)

visibilité, [Exemple](#)

visualiseur, [Cloud Objects Viewer](#)

CloudStorage, [Avec CloudStorage \(pour le cloud\)](#)CN1Lib, Le système de plug-in de

Codename One (CN1LIB)Cocher/décocher, [Quelques méthodes de MultiButton](#),

Exemple : Gérer le clic sur une case à cocher

Code natif, [Création d'un plug-in](#)

intégrer, Code natif (au-delà des limitations)

Code-barres, [Lecture de codes-barres et de QRcodes](#)CodeScanner, Lecture de codes-barres et de QRcodes

ComboBox, [ComboBox](#)Command, [Command](#), [Création d'un menu hamburger](#), Interface de recherche et d'affichage des films (FilmsPopulaires.java)

méthodes, [Quelques méthodes de Command](#)

CommonTransitions, [Transitions](#)Compilation, [La compilation avec Codename One](#), [La compilation avec Codename One](#), In-app purchase, Signature, compilation et déploiement

arguments, [Annexe 2 : Les arguments de compilation](#)

Compression/décompression, Des plug-ins gratuits à votre dispositionComScore, Des plug-ins gratuits à votre dispositionConnectionRequest, [Gestion de la connexion](#), Classe de la requête HTTP (RequeteReseau.java)

méthodes, [Quelques méthodes de ConnectionRequest](#)

Connexion

HTTP, [Réseau](#), Internet et services web

vitesse, Network Monitor (aide au débogage)

Contact, classe, [Création et accès aux contacts](#)Contacts, gestion des, [Création et accès aux contacts](#)ContactsManager, [Création et accès aux contacts](#)Container, Container, Interface de recherche et d'affichage des films (FilmsPopulaires.java)

créer, [Création d'un Container](#)

méthodes, [Quelques méthodes de Container](#)

CoordinateLayout, [CoordinateLayout](#)Couleur

arrière-plan, [Couleurs et images](#)

premier plan, [Couleurs et images](#)

Cryptographie, Des plug-ins gratuits à votre dispositionCSV, [Lecture d'un fichier CSV](#)

CSVParser, [Lecture d'un fichier CSV](#)Curseur

position, [Quelques méthodes de TextArea](#)

D

Database, [Avec Database](#) (pour les bases de données SQLite)

méthodes, [Quelques méthodes de Database](#)

[DataChangedListener](#), [Quelques méthodes de TextField](#), [Quelques méthodes de Slider](#), [Quelques méthodes de Calendar](#)[Déboguer](#), [Network Monitor \(aide au débogage\)](#), [Logging](#) [DefaultListModel](#), [Exemple : Créer un carrousel d'images](#)[Défilement](#)

image, [ImageViewer](#)

label, [Quelques méthodes de Label](#)

texte, [Quelques méthodes de Label](#)

transition, [Slide transition](#)

Dégradé, [Couleurs et images](#)[Déploiement](#), [Signature, compilation et déploiement](#)[Designer](#), [Présentation du Codename One Designer](#), [Exporter et importer depuis un autre fichier de ressources](#)[destroy\(\)](#), [Structure d'une application Codename One](#)[Diagrams](#), [Des plug-ins gratuits à votre disposition](#)[Dialog](#), [Dialog](#), [Avec NetworkManager et ConnectionRequest](#)

méthodes, [Quelques méthodes de Dialog](#)

Diaporama, [ImageViewer](#)[Display](#), [La classe Display](#)[Disposer](#)

à l'horizontale, [BoxLayout](#)

à la verticale, [BoxLayout](#)

avec des coordonnées, [CoordinateLayout](#)

contenu fluide, [FlowLayout](#)

selon une grille, [GridLayout](#)

sur les bords, [BorderLayout](#)

Données

base de, [Avec Database \(pour les bases de données SQLite\)](#), [Communication avec une base de données distante](#)

envoi, [Envoi de données \(upload\) avec MultipartRequest](#)

stockage, [Persistance des données](#)

téléchargement, [Téléchargement de données](#), [Utilisation d'un service web quelconque](#)

Download, [Téléchargement de données](#)

(voir aussi Télécharger)

Dropbox, [Des plug-ins gratuits à votre disposition](#)

E

E-mail, [Appel, e-mail, SMS](#)

Eclipse, [Sous Eclipse](#)

Écran

de veille, [La classe Display](#)
dimensions, [La classe Display](#)
orientation, [La classe Display](#)
verrouillage, [La classe Display](#)

Éditeur d'interfaces graphiques, [L'éditeur d'interfaces graphiques \(GUI Builder\)](#)
Éditeur graphique, [Hello Codename One](#), [Présentation du Codename One Designer](#)
[EDT](#), [Gestion de la connexion](#), [Présentation de l'EDT](#)

(voir aussi Event Dispatch Thread)

Effet visuel, [Transitions](#)
Emblème, [MultiButton](#), [Quelques méthodes de MultiButton](#)
Enregistrer du son, [Multimédia \(photo, audio, vidéo\)](#)
Évènement, [Exemple : Page avec titre et menus](#), [Gestion des événements](#)

clic, [Gestion de clic sur un bouton](#), [Exemple : Gérer le clic sur une case à cocher](#), [Exemple d'utilisation](#)

Event Dispatch Thread, [Dialog](#), [Gestion de la connexion](#), [Avec NetworkManager et ConnectionRequest](#), [Tirer et relâcher pour actualiser](#), [Présentation de l'EDT](#)

débogage, [Le menu Simulate](#)

Expressions régulières, [Des plug-ins gratuits à votre disposition](#)
Externalizable, [Exemples de stockage de données](#)

F

Fenêtre
créer, [Form](#)
d'attente, [Utilisation d'un service web quelconque](#), [Consommation du service web créé modale](#), [Dialog](#)
transparente, [Avec NetworkManager et ConnectionRequest](#)

Fichier

de ressources, [Hello Codename One](#), [Présentation du Codename One Designer](#)
manipulation, [Avec FileSystemStorage \(pour les fichiers\)](#)
parcourir, [Envoi de données \(upload\) avec MultipartRequest](#)
récupérer chemin, [Envoi de données \(upload\) avec MultipartRequest](#)

FileSystemStorage, [Avec FileSystemStorage \(pour les fichiers\)](#), [Avec NetworkManager et ConnectionRequest](#)

méthodes, [Quelques méthodes de FileSystemStorage](#)

FileTree, Envoi de données (upload) avec MultipartRequestFilmer, Multimédia (photo, audio, vidéo)FlowLayout, FlowLayoutFlurry Ads, Flurry AdsFlux RSS, RSSService (et RSSReader)Fondu enchaîné, Fade transitionFont, classe, Utilisation des polices TTF avec du codeForm, Hello Codename One, Form, Avec NetworkManager et ConnectionRequest, Avec ImageDownloadService, Utilisation d'un service web quelconque, La classe Display

créer, Création d'un Form
méthodes, Quelques méthodes de Form

Formulaire, FormFuseau horaire, Exemples : Calendriers avec différentes options

G

Géolocalisation, MapComponent
Gestionnaire de positionnement, Les layouts ou gestionnaires de positionnement (voir aussi Layout)

Google Analytics, Rapports statistiquesGoogle Maps, MapComponent, Des plug-ins gratuits à votre dispositionGoogle Mobile Ads, Google Mobile AdsGridLayout, GridLayoutGUI Builder, L'éditeur d'interfaces graphiques (GUI Builder)GZConnectionRequest, GZConnectionRequestGZipInputStream, GZConnectionRequestGZipOutputStream, GZConnectionRequest

H

Header HTTP, Quelques méthodes de ConnectionRequest
HTMLComponent, WebBrowser

I

Image
accès galerie, Accès à la galerie d'images et de vidéos
afficher, Label
arrière-plan, Couleurs et images
charger, Exemple : Afficher un texte court ou une image
diaporama, ImageViewer
fichier de ressources, Images
formats supportés, Images standards
résolution d'écran, Multi-images
SVG, Images SVG
téléchargement, Avec ImageDownloadService, Utilisation d'un service web quelconque

Image, classe, Exemple : Afficher un texte court ou une image ImageViewer, ImageViewer, Utilisation d'un service web quelconque

méthodes, Quelques méthodes de ImageViewer

In-App purchase, Le menu Simulate InfiniteProgress, Avec NetworkManager et ConnectionRequest, Avec certaines méthodes de la classe Util, Utilisation d'un service web quelconque, Consommation du service web créé init(), Hello Codename One, Classe principale (CN1FilmsBox.java) Inneractive, Inneractive Installation du plug-in, Téléchargement et installation du plug-in, Sous IntelliJ IDEA IntelliJ IDEA, Sous IntelliJ IDEA Interface graphique

avec ou sans éditeur graphique, Hello Codename One simulation, Hello Codename One

J

J2ME, Historique de Codename One

Jauge, Slider

JSON, Gestion de la connexion, Communication avec un service web, Utilisation d'un service web quelconque, Consommation du service web créé, Des plug-ins gratuits à votre disposition

JSONParser, Gestion de la connexion, Utilisation d'un service web quelconque, Consommation du service web créé

L

Label, Hello Codename One, Label, Avec ImageDownloadService

méthodes, Quelques méthodes de Label

Langue, Internationalisation/localisation, Interface de paramétrage de la langue (Parametres.java), Traduction de l'application LayeredLayout, LayeredLayout Layout, Les layouts ou gestionnaires de positionnement Lightweight, architecture, Pourquoi Codename One ? List, List, Avec ImageDownloadService, Utilisation d'un service web quelconque ListCellRenderer, Quelques méthodes d'AutoCompleteTextField, Création d'une liste complexe (Personnalisation du rendu) Liste

déroulante, ComboBox, ZooZ

élaborée, Crédit d'une liste complexe (Personnalisation du rendu)

orientation, Quelques méthodes de List

premier élément, Quelques méthodes de List

simple, Crédit d'une liste simple

Localisation, Internationalisation/localisation, Interface de paramétrage de la langue

(Parametres.java), Traduction de l'application LocationManager, Exemple : Localiser l'utilisateur sur OpenStreetMap Log, classe, Logging LWUIT, Historique de Codename One

M

Map, Utilisation d'un service web quelconque, Consommation du service web créé MapComponent, MapComponent, Des plug-ins gratuits à votre disposition méthodes, Quelques méthodes usuelles de MapComponent

Media, classe, Lecture, Exemple : Lire un fichier audio ou vidéo, Avec NetworkManager et ConnectionRequest MediaManager, Lecture, Exemple : Lire un fichier audio ou vidéo MediaPlayer, Le composant MediaPlayerMenu, Command

hamburger, Toolbar, Menu hamburger, Les constantes de thèmes du menu hamburger (side menu), Interface de recherche et d'affichage des films (FilmsPopulaires.java)

Message, classe, Appel, e-mail, SMS Mise en pause, Hello Codename One Motion, Transitions Multi-image, Multi-images, Interface principale (Accueil.java) MultiButton, MultiButton, Classe de la requête HTTP (RequeteReseau.java)

méthodes, Quelques méthodes de MultiButton

MultipartRequest, Envoi de données (upload) avec MultipartRequest

N

NativeInterface, Code natif (au-delà des limitations)

Navigateur web, WebBrowser

NetBeans, Sous NetBeans

Network Monitor, Network Monitor (aide au débogage)

NetworkManager, Gestion de la connexion

méthodes, Quelques méthodes de NetworkManager

Notification push, Notification push

O

Objet

CloudObject, Avec CloudStorage (pour le cloud)

sérialiser, Exemples de stockage de données

stocker, Exemples de stockage de données

Onglet, TabsOnOffSwitch, OnOffSwitchOpenStreetMap, MapComponentOrientation de l'écran, La classe Display

P

Page
créer, Form
récupération après pause, Hello Codename One

Paiement, Effectuer un paiement depuis une applicationPhotographier, Multimédia (photo, audio, vidéo)Picker, PickerPlug-in, Le système de plug-in de Codename One (CN1LIB), Création d'un plug-in

créer, Création d'un plug-in
utiliser dans un projet, Utilisation d'un plug-in

PointsLayer, Exemple : Localiser l'utilisateur sur OpenStreetMapPolice de caractères, Polices de caractèresPreferences, classe, Avec PreferencesProgress, Avec NetworkManager et ConnectionRequestProjet

créer, Hello Codename One
exécuter, Hello Codename One

Publicité, Monétisation des applications gratuites, Flurry AdsPubNub, Des plug-ins gratuits à votre dispositionPull to refresh, Tirer et relâcher pour actualiser, Interface de recherche et d'affichage des films (FilmsPopulaires.java)Push, notification, Notification push, Envoi de messages depuis une application web

Q

QRcode, Lecture de codes-barres et de QRcodes

R

RadioButton, RadioButton
méthodes, Quelques méthodes de RadioButton

Recouvrement, effet de, Cover et Uncover transitionRequête, Interface de recherche et d'affichage des films (FilmsPopulaires.java), Classe de la requête HTTP (RequeteReseau.java)

de connexion, Gestion de la connexion

de type multipart, [Envoi de données \(upload\) avec MultipartRequest](#)
[décompresser, GZConnectionRequest](#)
[stopper, Quelques méthodes de NetworkManager](#)
temps d'exécution, [Quelques méthodes de ConnectionRequest](#)

Réseau, [Gestion de la connexion](#)
[Résolutions d'écran, Multi-images](#)
[Ressources, fichier de, Hello Codename One, Présentation du Codename One Designer, Fichiers divers](#)
protection, [Protection par mot de passe](#)

REST, [Communication avec une base de données distante](#)
[Restauration après mise en pause, Hello Codename One](#)
[RSSReader, RSSService \(et RSSReader\)](#)
[RSSService, RSSService \(et RSSReader\)](#)
[Runnable, Lire un fichier en boucle, La classe Display](#)

S

Sélecteur de données, [Picker](#)
[SelectionListener, Quelques méthodes de Tabs](#)
[Sérialisation, Exemples de stockage de données](#)
Service web, [Communication avec un service web, Brève analyse de l'application à concevoir](#)
[ShareButton, ShareButton, Interface principale \(Accueil.java\)](#)
méthodes, [Quelques méthodes de ShareButton](#)

Side menu (voir Menu hamburger)
[Signature, Annexe 3 : Signature d'une application, Sous Windows Phone](#)
[Simulateur, Hello Codename One, Présentation et fonctionnement du simulateur, Le menu Skins](#)

capture d'écran, [Le menu Simulate](#)

Slider, [Slider](#)

méthodes, [Quelques méthodes de Slider](#)

SMS, [Appel, e-mail, SMS](#)
[Socket, Des plug-ins gratuits à votre disposition](#)
[SpanButton, SpanButton](#)
[SpanLabel, SpanLabel](#)
[SQLite, Avec Database \(pour les bases de données SQLite\)](#)

chemin sous Windows, [Quelques méthodes de Database](#)

[start\(\), Structure d'une application Codename One, Hello Codename One, Classe principale \(CN1FilmsBox.java\)](#)
[Statistiques, Flurry Ads, Rapports statistiques](#)
[Stockage, Persistance des données, Quelques méthodes de CloudStorage](#)

base de données, [Avec Database \(pour les bases de données SQLite\)](#)
cloud, [Avec CloudStorage \(pour le cloud\)](#)

des préférences, [Avec Preferences](#)
fichiers, [Avec FileSystemStorage](#) (pour les fichiers)
polyvalent, [Avec Storage](#)
récupérer chemin, [Avec NetworkManager et ConnectionRequest](#)
RMS, [Avec Storage](#)

`stop()`, [Structure d'une application Codename One](#), [Hello Codename OneStorage](#), [Le menu Simulate](#), [Avec Storage](#), [Avec ImageDownloadService](#)

lecture de données, [Exemples de lecture de données](#)
méthodes, [Quelques méthodes de Storage](#)

Style, classe, [Styles d'un composant](#), [Les thèmes](#), [Utilisation des polices TTF avec du code Superposition](#), [LayeredLayout](#)
[SwipeableContainer](#), [SwipeableContainer](#)

T

Tabs, [Tabs](#), [Interface de recherche et d'affichage des films \(FilmsPopulaires.java\)](#)
méthodes, [Quelques méthodes de Tabs](#)

Télécharger

données, [Téléchargement de données](#)
images, [Avec ImageDownloadService](#)

TextArea, [TextArea](#)

méthodes, [Quelques méthodes de TextArea](#)

Texte

afficher, [Label](#)
défilement, [Quelques méthodes de Label](#)
éditable, [Quelques méthodes de TextArea](#)
liste, [Création d'une liste complexe \(Personnalisation du rendu\)](#)
long, [SpanLabel](#)
par défaut, [Quelques méthodes de TextArea](#)
zone multiligne, [TextArea](#)

TextField, [TextField](#), [Exemple d'utilisation](#), [Consommation du service web créé](#), [Interface de recherche et d'affichage des films \(FilmsPopulaires.java\)](#)

méthodes, [Quelques méthodes de TextField](#)

Thème, [Les thèmes](#), [Les constantes de thèmes](#)

constante de, [Les constantes de thèmes](#)
natif, [Hello Codename One](#)

Thread, [Gestion de la connexion](#), [La classe Display](#), [Tirer et relâcher pour actualiser](#),
[Présentation de l'EDT](#)

arrêter, [Quelques méthodes de NetworkManager](#)
en cours d'exécution, [Quelques méthodes de NetworkManager](#)
principal, [La classe Display](#)
réseau, [Avec NetworkManager et ConnectionRequest](#)

TimeZone, [Exemples : Calendriers avec différentes options](#)
Titre de l'application, [FormToast](#), [Des plug-ins gratuits à votre disposition](#), [Utilisation d'un plug-in](#)
[Toolbar](#), [Toolbar](#), [Interface de recherche et d'affichage des films \(FilmsPopulaires.java\)](#)
[Traduction](#), [Internationalisation/localisation](#), [Interface de paramétrage de la langue \(Parametres.java\)](#),
[Traduction de l'application](#)
[Transition](#), [Transitions](#), [Utilisation d'un service web quelconque](#), [Les constantes de thèmes](#)

(un)cover, [Cover et Uncover transition](#)
fade, [Fade transition](#)
slide, [Slide transition](#)

Transparence, [Couleurs et images](#)
Type

externalisable, [Exemples de stockage de données](#)
multipart, [Envoi de données \(upload\) avec MultipartRequest](#)

U

UIID, [Concept de l'UIID](#)
[Upload](#), [Envoi de données \(upload\) avec MultipartRequest](#)
User Interface Identifier, [Concept de l'UIID](#)
(voir aussi UIID)

Util, classe, [Avec NetworkManager et ConnectionRequest](#), [Avec certaines méthodes de la classe Util](#)

V

Vector, [Exemple : Liste déroulante](#), [Exemples de stockage de données](#), [RSSService \(et RSSReader\)](#), [Utilisation d'un service web quelconque](#)
Veille de l'écran, [La classe Display](#)
Verrouillage de l'écran, [La classe Display](#)
Vibreur, [La classe Display](#)

Vserv, [Vserv](#)

W

WebBrowser, [WebBrowser](#)
méthodes, [Quelques méthodes de WebBrowser](#)

X

XML, [Gestion de la connexion](#), [Communication avec un service web](#), [Lecture d'un fichier XML](#)
[XMLParser](#), [Gestion de la connexion](#)

Z

Zoom, [ImageViewer](#), Exemple : Localiser l'utilisateur sur OpenStreetMap
[ZooZ](#), [ZooZ](#)

- [Codename One - Développer en Java pour iOS, Android, BlackBerry et Windows Phone](#)
 - [Copyright](#)
 - [À propos de l'auteur](#)
 - [Préface](#)
 - [Avant-propos](#)
 - [Public visé et prérequis](#)
 - [Réglage de la largeur de l'écran](#)
 - [Sources des exemples](#)
 - [Accès aux vidéos](#)
 - [URL raccourcies](#)
 - [Remerciements](#)
 - [Introduction](#)
 - [Pourquoi utiliser un outil multiplateforme pour la programmation mobile ?](#)
 - [Historique de Codename One](#)
 - [Pourquoi Codename One ?](#)
 - [Avantages](#)
 - [Inconvénients](#)
 - [Démarrage](#)
 - [Téléchargement et installation du plug-in](#)
 - [Sous NetBeans](#)
 - [Sous Eclipse](#)
 - [Sous IntelliJ IDEA](#)
 - [Structure d'une application Codename One](#)
 - [Hello Codename One](#)
 - [La compilation avec Codename One](#)
 - [Présentation et fonctionnement du simulateur](#)
 - [Le menu Simulate](#)
 - [Le menu Skins](#)
 - [Le menu Help](#)
 - [Les composants graphiques](#)
 - [Les conteneurs principaux](#)
 - [Container](#)
 - [Création d'un Container](#)
 - [Quelques méthodes de Container](#)
 - [Form](#)
 - [Création d'un Form](#)
 - [Quelques méthodes de Form](#)
 - [Dialog](#)
 - [Création d'un Dialog](#)
 - [Quelques méthodes de Dialog](#)
 - [Exemple : Boîtes de dialogue sans instanciation manuelle de la classe Dialog](#)
 - [Exemple : Boîte de dialogue personnalisée avec instantiation](#)
 - [Tabs](#)

- [Création d'un Tabs](#)
- [Quelques méthodes de Tabs](#)
- [Exemple : Interface avec onglets](#)
- [Les layouts ou gestionnaires de positionnement](#)
 - [BoxLayout](#)
 - [Création d'un BoxLayout](#)
 - [Quelques méthodes de BoxLayout](#)
 - [Exemple : Positionner à la verticale ou à l'horizontale](#)
 - [BorderLayout](#)
 - [Création d'un BorderLayout](#)
 - [Quelques méthodes de BorderLayout](#)
 - [Exemple : Positionner au centre et à la périphérie](#)
 - [FlowLayout](#)
 - [Création d'un FlowLayout](#)
 - [Quelques méthodes de FlowLayout](#)
 - [Exemple : Contenus fluides avec différents alignements](#)
 - [GridLayout](#)
 - [Création d'un GridLayout](#)
 - [Quelques méthodes de GridLayout](#)
 - [Exemple : Positionner selon une grille](#)
 - [LayeredLayout](#)
 - [Création d'un LayeredLayout](#)
 - [CoordinateLayout](#)
 - [Création d'un CoordinateLayout](#)
 - [Exemple : Positionner avec des coordonnées](#)
- [Autres composants](#)
 - [Command](#)
 - [Exemple : Page avec titre et menus](#)
 - [Quelques méthodes de Command](#)
 - [Label](#)
 - [Exemple : Afficher un texte court ou une image](#)
 - [Quelques méthodes de Label](#)
 - [SpanLabel](#)
 - [Exemple : Afficher un texte long](#)
 - [Button](#)
 - [Exemple : Créer un bouton simple](#)
 - [Gestion de clic sur un bouton](#)
 - [Quelques méthodes de Button](#)
 - [SpanButton](#)
 - [Exemple : Bouton avec un texte long](#)
 - [MultiButton](#)
 - [Exemple : Bouton complexe](#)
 - [Quelques méthodes de MultiButton](#)
 - [CheckBox](#)
 - [Exemple : Cases à cocher avec texte et/ou image](#)
 - [Exemple : Gérer le clic sur une case à cocher](#)

- [Quelques méthodes de CheckBox](#)
- [RadioButton](#)
 - [Exemple : Cases à cocher à choix unique](#)
 - [Quelques méthodes de RadioButton](#)
 - [Quelques méthodes de ButtonGroup](#)
- [TextArea et TextField](#)
 - [TextArea](#)
 - [Exemple : Zones de saisie avec ou sans contrainte](#)
 - [Quelques méthodes de TextArea](#)
 - [TextField](#)
 - [Exemple : Zone de saisie d'une seule ligne](#)
 - [Quelques méthodes de TextField](#)
 - [Contraintes de saisie](#)
- [AutoCompleteTextField](#)
 - [Exemple : Zone de saisie avec autocomplétion](#)
 - [Quelques méthodes d'AutoCompleteTextField](#)
- [ComboBox](#)
 - [Exemple : Liste déroulante](#)
 - [Quelques méthodes de ComboBox](#)
- [Slider](#)
 - [Exemple : Jauge interactive](#)
 - [Quelques méthodes de Slider](#)
- [Calendar](#)
 - [Exemples : Calendriers avec différentes options](#)
 - [Quelques méthodes de Calendar](#)
- [WebBrowser](#)
 - [Exemple : Lire une page HTML locale ou distante](#)
 - [Quelques méthodes de WebBrowser](#)
- [Picker](#)
 - [Exemple : Sélecteurs de date et/ou heure](#)
 - [Quelques méthodes de Picker](#)
- [OnOffSwitch](#)
 - [Exemple : Bouton Marche/Arrêt](#)
 - [Quelques méthodes de OnOffSwitch](#)
- [ShareButton](#)
 - [Exemple : Bouton de partage de données](#)
 - [Quelques méthodes de ShareButton](#)
- [ImageViewer](#)
 - [Exemple : Créer un carrousel d'images](#)
 - [Quelques méthodes de ImageViewer](#)
- [MapComponent](#)
 - [Exemple : Localiser l'utilisateur sur OpenStreetMap](#)
 - [Quelques méthodes usuelles de MapComponent](#)
- [List](#)
 - [Création d'une liste simple](#)
 - [Création d'une liste complexe \(Personnalisation du rendu\)](#)

- [Quelques méthodes de List](#)
- [SwipeableContainer](#)
 - [Quelques méthodes de SwipeableContainer](#)
- [Toolbar](#)
 - [Quelques méthodes de Toolbar](#)
- [Styles et transitions](#)
 - [Styles d'un composant](#)
 - [Couleurs et images](#)
 - [Caractères et polices de caractères](#)
 - [Alignment, marges, bordures et autres](#)
 - [Transitions](#)
 - [Slide transition](#)
 - [Fade transition](#)
 - [Cover et Uncover transition](#)
 - [Flip transition](#)
 - [Exemple : Effets de transition entre quatre pages](#)
- [Persistance des données](#)
 - [Avec Storage](#)
 - [Exemples de stockage de données](#)
 - [Exemples de lecture de données](#)
 - [Quelques méthodes de Storage](#)
 - [Avec Preferences](#)
 - [Exemple : Stockage et lecture](#)
 - [Quelques méthodes de Preferences](#)
 - [Avec Database \(pour les bases de données SQLite\)](#)
 - [Requête simple](#)
 - [Exemple d'utilisation](#)
 - [Quelques méthodes de Database](#)
 - [Avec FileSystemStorage \(pour les fichiers\)](#)
 - [Exemple](#)
 - [Quelques méthodes de FileSystemStorage](#)
 - [Avec CloudStorage \(pour le cloud\)](#)
 - [Exemple](#)
 - [Cloud Objects Viewer](#)
 - [Quelques méthodes de CloudStorage](#)
- [Multimédia \(photo, audio, vidéo\)](#)
 - [Capture](#)
 - [Exemple : Créer des boutons d'enregistrement](#)
 - [Quelques méthodes de Capture](#)
 - [Lecture](#)
 - [Exemple : Lire un fichier audio ou vidéo](#)
 - [Lire un fichier audio ou vidéo depuis Internet](#)
 - [Lire un fichier en boucle](#)
 - [Quelques méthodes de Media](#)
 - [Le composant MediaPlayer](#)
 - [Exemple : Interface avec lecteur multimédia intégré](#)

- [Quelques méthodes de MediaPlayer](#)
 - [Accès à la galerie d'images et de vidéos](#)
- [Réseau, Internet et services web](#)
 - [Gestion de la connexion](#)
 - [Téléchargement de données](#)
 - [Avec NetworkManager et ConnectionRequest](#)
 - [Avec ImageDownloadService](#)
 - [Avec URLImage](#)
 - [Avec certaines méthodes de la classe Util](#)
 - [Envoi de données \(upload\) avec MultipartRequest](#)
 - [Autres classes dérivées de ConnectionRequest](#)
 - [RSSService \(et RSSReader\)](#)
 - [GZConnectionRequest](#)
 - [Quelques méthodes de ConnectionRequest](#)
 - [Quelques méthodes de NetworkManager](#)
 - [Communication avec un service web](#)
 - [Utilisation d'un service web quelconque](#)
 - [Communication avec une base de données distante](#)
 - [Création d'un simple service web REST en PHP](#)
 - [Consommation du service web créé](#)
 - [Network Monitor \(aide au débogage\)](#)
- [Codename One Designer](#)
 - [Présentation du Codename One Designer](#)
 - [Images](#)
 - [Images standards](#)
 - [Ajouter une image](#)
 - [Récupérer une image depuis un fichier de ressources](#)
 - [Multi-images](#)
 - [Images SVG](#)
 - [Images non utilisées et taille des images](#)
 - [Fichiers divers](#)
 - [Les thèmes](#)
 - [Utilisation des thèmes existants](#)
 - [Concept de l'UIID](#)
 - [Bases de la création d'un thème](#)
 - [Les constantes de thèmes](#)
 - [L'éditeur d'interfaces graphiques \(GUI Builder\)](#)
 - [Présentation de l'interface et fonctionnement du GUI Builder](#)
 - [Création d'interfaces graphiques](#)
 - [Gestion des événements](#)
 - [Codename One LIVE et prévisualisation en live \(Live Preview\)](#)
 - [Police de caractères](#)
 - [Utilisation des polices TTF avec du code](#)
 - [Utilisation des polices TTF avec le Designer](#)
 - [Internationalisation/localisation](#)
 - [Fonctionnalités diverses](#)

- [Protection par mot de passe](#)
 - [Exporter et importer depuis un autre fichier de ressources](#)
- [Plug-ins et code natif](#)
 - [Le système de plug-in de Codename One \(CN1LIB\)](#)
 - [Des plug-ins gratuits à votre disposition](#)
 - [Utilisation d'un plug-in](#)
 - [Création d'un plug-in](#)
 - [Code natif \(au-delà des limitations\)](#)
- [Monétisation](#)
 - [Monétisation des applications gratuites](#)
 - [Google Mobile Ads](#)
 - [Inneractive](#)
 - [Vserv](#)
 - [Flurry Ads](#)
 - [Effectuer un paiement depuis une application](#)
 - [ZooZ](#)
 - [In-app purchase](#)
 - [Quelques méthodes de Purchase](#)
- [Fonctionnalités diverses](#)
 - [La classe Display](#)
 - [Appel, e-mail, SMS](#)
 - [Lecture de codes-barres et de QRcodes](#)
 - [Rapports statistiques](#)
 - [Logging](#)
 - [Menu hamburger](#)
 - [Création d'un menu hamburger](#)
 - [Modification de l'icône par défaut](#)
 - [Changement de l'emplacement du menu](#)
 - [Ajout d'un composant quelconque](#)
 - [Personnalisation de la couleur ou de l'image de fond](#)
 - [Les constantes de thèmes du menu hamburger \(side menu\)](#)
 - [Tirer et relâcher pour actualiser](#)
 - [Lecture d'un fichier CSV](#)
 - [Lecture d'un fichier XML](#)
 - [Création et accès aux contacts](#)
 - [Ajout d'un nouveau contact](#)
 - [Lecture d'un ou de plusieurs contacts](#)
 - [Quelques méthodes de ContactsManager](#)
 - [Notification push](#)
 - [Mise en place de la partie cliente](#)
 - [Envoi de messages depuis une application mobile](#)
 - [Envoi de messages depuis une application web](#)
- [Étude de cas : Création d'une application de A à Z](#)
 - [Brève analyse de l'application à concevoir](#)
 - [Les classes du projet](#)
 - [Création de la classe principale et des interfaces graphiques](#)

- [Interface principale \(Accueil.java\)](#)
- [Interface de recherche et d'affichage des films \(FilmsPopulaires.java\)](#)
- [Interface de paramétrage de la langue \(Parametres.java\)](#)
- [Implémentation des fonctionnalités](#)
 - [Interface principale \(Accueil.java\)](#)
 - [Interface de recherche et d'affichage des films \(FilmsPopulaires.java\)](#)
 - [Interface de paramétrage de la langue \(Parametres.java\)](#)
 - [Classe de la requête HTTP \(RequeteReseau.java\)](#)
 - [Traduction de l'application](#)
 - [Classe principale \(CN1FilmsBox.java\)](#)
- [Idées d'extension de l'application](#)
- [Signature, compilation et déploiement](#)
- [Conclusion](#)
 - [Contribuer à la conception de Codename One](#)
 - [Contribuer à la galerie d'applications](#)
 - [Ressources complémentaires](#)
- [Annexe 1 : Event Dispatch Thread \(EDT\)](#)
 - [Présentation de l'EDT](#)
 - [Exécution d'une action dans l'EDT depuis un autre thread \(Call-Serially/CallSeriallyAndWait\)](#)
 - [Exécution d'une action dans un autre thread depuis l'EDT \(InvokeAndBlock\)](#)
 - [Détection des violations de l'EDT](#)
- [Annexe 2 : Les arguments de compilation](#)
 - [Tableaux récapitulatifs des arguments de compilation](#)
- [Annexe 3 : Signature d'une application](#)
 - [Sous Android](#)
 - [Sous iOS](#)
 - [Sous Blackberry](#)
 - [Sous Windows Phone](#)
- [Index](#)