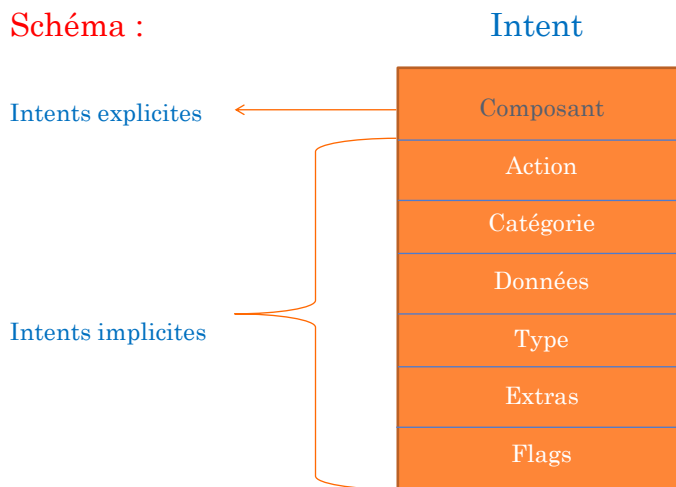


# INTENTS

Un mécanisme qui permet de faire exécuter certaines actions et de faire circuler des messages entre applications ou à l'intérieur d'une même application.

Schéma :



Pour qu'un intent soit dit « explicite », il suffit que son champ composant soit renseigné.

```
Intent i = new Intent(this, ActivityDeux.class);
```

Ce champ permet de définir le destinataire de l'intent, celui qui devra le gérer. Ce champ est constitué de deux informations : le *package* où se situe le composant, ainsi que le *nom* du composant.

Ainsi, quand l'intent sera exécuté, Android pourra retrouver le composant de destination de manière précise.

3

Pour les intents « implicites » on ne connaît pas de manière précise le destinataire de l'intent, c'est pourquoi on va s'appliquer à renseigner d'autres champs pour laisser Android déterminer qui est capable de réceptionner cet intent.

```
Intent i = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.bibi.com"));
```

Il faut au moins fournir deux informations essentielles :

**Une action :** ce qu'on désire que le destinataire fasse.

**Un ensemble de données :** sur quelles données le destinataire doit effectuer son action.

4

Il existe aussi d'autres informations, pas forcément obligatoires, mais qui ont aussi leur utilité propre le moment venu :

**La catégorie :** permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.

**Le type :** pour indiquer quel est le type des données incluses. Normalement ce type est contenu dans les données, mais en précisant cet attribut vous pouvez désactiver cette vérification automatique et imposer un type particulier.

5

**Les extras :** pour ajouter du contenu à vos intents afin de les faire circuler entre les composants.

**Les flags :** permettent de modifier le comportement de l'intent.

6

## Cas 1 : Intents explicites

Un Intent explicite définit explicitement le composant qui doit être appelé par le système Android, en utilisant la classe Java comme identifiant.

### Exemple :

```
Intent leIntent = new Intent(Activite_de_depart.this,
    Activite_de_destination.class);
```

7

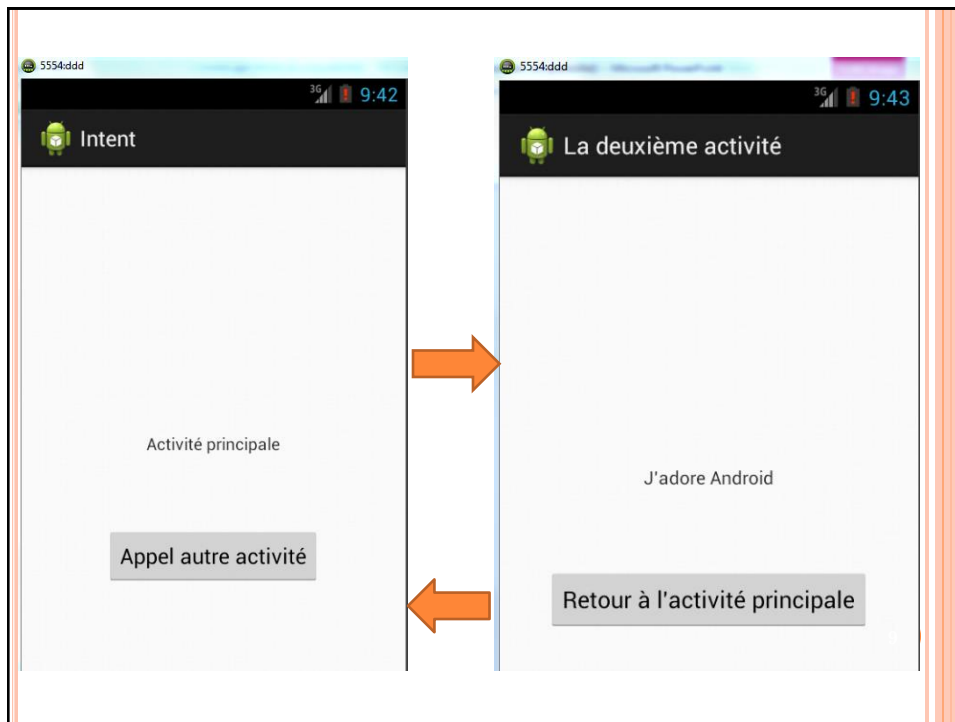
Il existe ensuite deux façons de lancer l'intent, selon qu'on veuille que le composant de destination nous renvoie une réponse ou pas.

### Sans retour

Si vous ne vous attendez pas à ce que la nouvelle activité vous renvoie un résultat, alors vous pouvez l'appeler très naturellement avec `void startActivity (Intent leIntent)` dans votre activité.

La nouvelle activité sera indépendante de l'actuelle. Elle entreprendra un cycle d'activité normal, c'est-à-dire en commençant par un `onCreate`.

8



## Projet Intent

### MainActivity.java

@Override

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button b = (Button)findViewById(R.id.button1);
```

```
b.setOnClickListener(new OnClickListener() {
```

@Override

```
public void onClick(View v) {
    Intent i = new Intent(MainActivity.this, DeuxiemeActivite.class);
    i.putExtra("Valeur1", "J'adore Android");
    i.putExtra("Valeur2", "Je commence à comprendre les intents");
    startActivity(i);
} }); }
```

## DeuxiemeActivite.java

```
public class DeuxiemeActivite extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.deuxieme_activite);
        // getIntent() est une méthode de Activity
        Bundle extras = getIntent().getExtras();
        String inputString = extras.getString("Valeur1");
        TextView view = (TextView) findViewById(R.id.elem);
        view.setText(inputString);
        Button b = (Button)findViewById(R.id.button2);
        b.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                DeuxiemeActivite.this.finish();
            }
        });
    }
}
```

11

## Layout de la première activité

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/elem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/elem"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        android:text="Appel autre activité" />

</RelativeLayout>
```

12

## Layout de la 2<sup>ème</sup> activité

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/elem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/elem"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        android:text="Retour à l'activité principale" />

</RelativeLayout>
```

13

## Exercice 1a :

Concevoir une application comportant 2 activités.

### Activité 1

Comporte 3 champs :

- Nom
- Prénom
- Âge
- Sexe (F ou M)



### Activité 2

Reçoit les valeurs des 4 champs envoyés par l'activité 1 et les affiche selon le format :

NOM = xxx  
 PRÉNOM = xxx  
 ÂGE = xxx  
 SEXE= Féminin ou Masculin selon le cas.

14

**Nota :** Il faut préciser dans le Manifest que vous avez désormais deux activités au sein de votre application .

```
<activity
    android:name="com.example.intent.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:label="La deuxième activité"
    android:name=".DeuxiemeActivite" >
</activity>
```

15

## Avec retour

Cette fois, on veut qu'au retour de l'activité qui vient d'être appelée cette dernière nous renvoie un petit *feedback*.

Pour cela, on utilisera la méthode **void** **startActivityForResult(Intent intent, int requestCode)**, avec **requestCode** un code passé qui permet d'identifier de manière unique un intent.

Ce code doit être supérieur ou égal à 0, sinon Android considérera que vous n'avez pas demandé de résultat.

16



Quand l'activité appelée s'arrêtera, la première méthode de *callback* appelée dans l'activité précédente sera **void onActivityResult(int requestCode, int resultCode, Intent data)**.

On retrouve **requestCode**, qui sera le même code que celui passé dans le **startActivityForResult** et qui permet de repérer quel intent a provoqué l'appel de l'activité dont le cycle vient de s'interrompre.

**resultCode** est quant à lui un code renvoyé par l'activité qui indique comment elle s'est terminée (typiquement **Activity.RESULT\_OK** si l'activité s'est terminée normalement, ou **Activity.RESULT\_CANCELED** s'il y a eu un problème ou qu'aucun code de retour n'a été précisé). Enfin, **intent** est un intent qui contient éventuellement des données.

17

## Dans la première activité, l'appel :

```
public final static int CODE = 0;
public final static String REPONSE =
"com.example.intent.REPONSE";
.....
public void onClick(View v) {
    Intent i = new Intent(MainActivity.this,
deuxiemeActivite.class);
    // On associe l'identifiant à notre intent
    startActivityForResult(i, CODE);
}
```

18

Reception des résultats de deuxiemeActivite :

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
```

```
    // On vérifie tout d'abord à quel intent on fait référence ici à l'aide de
    notre identifiant
```

```
    if (requestCode == CODE) {
```

```
        // On vérifie aussi que l'opération s'est bien déroulée
```

```
        if (resultCode == RESULT_OK) {
```

```
            // On affiche le bouton qui a été choisi
```

```
            Toast.makeText(this, "Vous avez choisi le bouton " +
data.getStringExtra(REPONSE),
Toast.LENGTH_SHORT).show();
```

```
        }
```

```
    }
```

19

## Dans la deuxiemeActivite :

```
@Override
```

```
    public void onClick(View v) {
```

```
        Intent result = new Intent();
```

```
        result.putExtra(MainActivity.REPONSE, "1");
```

```
        setResult(RESULT_OK, result);
```

```
        finish();
```

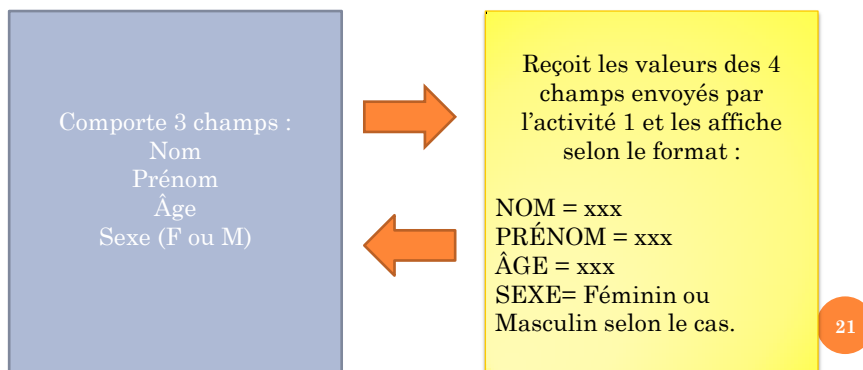
```
    }
```

20

## Exercice 1b:

Reprendre l'exercice précédent, mais cette fois-ci l'activité 2 va retourner :

- **Adulte** si l'âge est  $\geq 18$ , **Adolescent** si l'âge est comprise entre 12 et 17 et **Enfant** si l'âge est  $< 12$ .



21

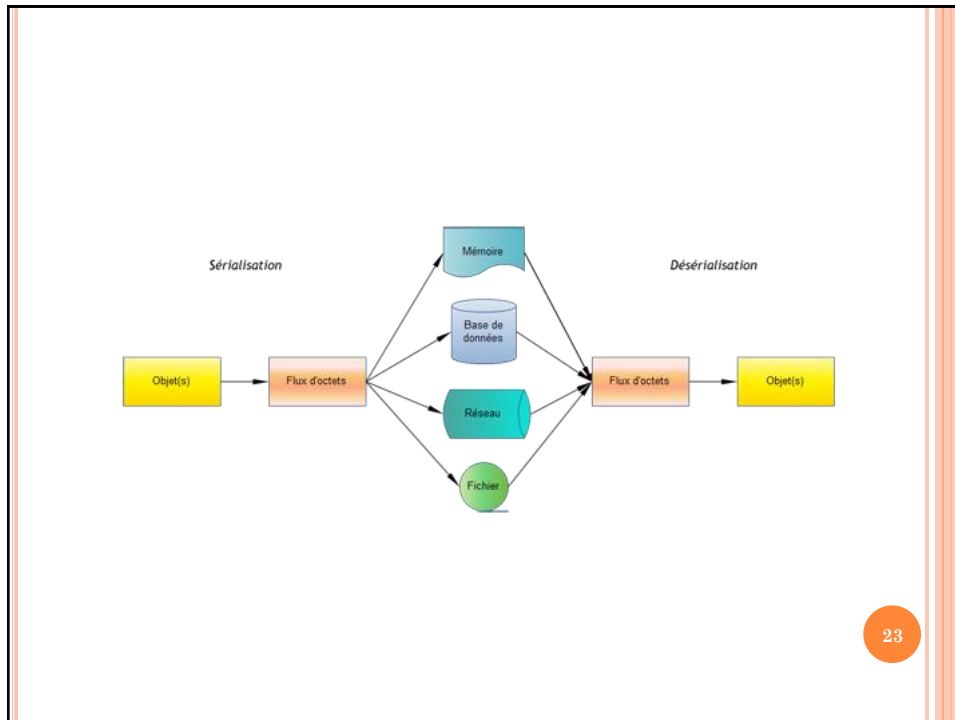
## La sérialisation

### Révision Java

La sérialisation est un procédé introduit dans le JDK version 1.1 qui permet de rendre un objet ou un graphe d'objets de la JVM persistant pour stockage ou échange et vice versa.

Cet objet est mis sous une forme sous laquelle il pourra être reconstitué à l'identique. Ainsi il pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans une autre JVM.

22



23

Java fournit un format standard pour la sérialisation.

Il est de ce fait inutile de créer un format particulier pour sauvegarder et relire un objet.

Le format utilisé est indépendant du système d'exploitation.

Ainsi, un objet sérialisé sur un système peut être réutilisé par un autre système pour recréer l'objet.

24

Il existe plusieurs formats de sérialisation appartenant à deux grandes familles :

**formats binaires** : c'est le format par défaut

**formats textes** : ils sont plus portables car ils utilisent généralement une structuration standard (XML, JSON, ...), peuvent être facilement modifiés et consomment plus de ressources pour être traités

25

#### Exemple de fichier XML

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
```

26

```

<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="web" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>

```

27

## Exemple de JSON

```

var automobiles = {
  "Acura": ["MDX", "RDX", "TL", "RL" ],
  "Audi": ["A4", "A6", "A8", "S4", "S6" ],
  "BMW": ["M3", "M5", "M6", "Z4" ],
  "Chevrolet": ["Aveo", "Cobalt", "Colorado", "Corvette",
  "Equinox" ],
  "Chrysler": ["300", "Aspen", "PT Cruser", "Sebring" ],
  "Ford": ["Edge", "Escape", "Fusion", "Focus" ],
  "Honda": ["Accord", "Civic", "Element", "Fit" ],
  "Hyundai": ["Accent", "Elantra", "Tiburon", "Sonata",
  "Tucson" ]
};

```

28

## JSON ET XML

```
{
  "nom": "Bob",
  "age": 34,
  "adresse": { "rue": "avenue
Grande", "ville": "Rio", "code": 86945 },
  "telephone": [ { "type": "maison",
"numero" : 123456 },
{ "type": "portable", "numero":
654321 } ]
}
```

L'équivalent en XML serait :

```
<utilisateur nom="Bob" age="34">
  <adresse>
    <rue>avenue Grande</rue>
    <ville>Rio</ville>
    <code>86945</code>
  </adresse>
  <telephones>
    <telephone type="maison">123456</telephone>
    <telephone
type="portable">654321</telephone>
  </telephones>
</utilisateur>
```

29

## Tous les objets ne sont pas sérialisable :

Généralement ce sont des objets qui ont des références sur des éléments du système d'exploitation (threads, fichiers, ...). Coûts implantation.

Pour pouvoir être sérialisée, une classe doit implémenter l'interface **java.io.Serializable**.

30

### Exemple : écriture/lecture d'objets

```
import java.util.*;
import java.io.*;

public class EcrireObjet
{
    public static void main (String args []) throws IOException
    {
        String chaine = null;
        Vector tabElements = new Vector();
        String unJeton = null;
```

31

```
BufferedReader clavier = new BufferedReader (new
InputStreamReader(System.in));
```

```
    System.out.println("Entrer votre suite
d'entiers sur la prochaine ligne");
```

```
    chaine = clavier.readLine();
```

```
System.out.println();
```

```
    System.out.println("chaine lue : " + chaine);
```

32



```
StringTokenizer st=new StringTokenizer(chaine, " ");

while (st.hasMoreTokens())
{
    unJeton = st.nextToken();
    tabElements.addElement(new Integer
        (Integer.parseInt(unJeton)));
    System.out.println("jeton : " + unJeton);
}
System.out.println("contenu du vecteur " +
tabElements);
```

33

```
ObjectOutputStream fichier = new ObjectOutputStream
(new FileOutputStream ("d:\\fichierObjet.txt"));

for (int i = 0; i < tabElements.size(); i++)
{
    fichier.writeObject(tabElements.elementAt(i));
    System.out.println(tabElements.elementAt(i));
}
fichier.close();
```

34

```

try {
    ObjectInputStream fichierIn = new ObjectInputStream
(new FileInputStream ("d:\\fichierObjet.txt"));
    System.out.println("lecture à partir d'un fichier
d'objets");
    for (int i = 0; i < tabElements.size(); i++)
    {
        System.out.println(fichierIn.readObject());
    }
    fichierIn.close();
    System.out.println("Fin d'execution");
} catch (FileNotFoundException e)
{ System.out.println("Fichier 'octet.txt' n'existe pas");
}

```

35

```

catch (IOException e)
{System.out.println("Erreur d'entree/sortie de
fichier");
}
}
}

```

Entrer votre suite d'entiers sur la prochaine ligne  
34 45 56 78

chaîne lue : 34 45 56 78  
jeton : 34  
jeton : 45  
jeton : 56  
jeton : 78  
contenu du vecteur [34, 45, 56, 78]  
34  
45  
56  
78  
lecture à partir d'un fichier d'objets  
34  
45  
56  
78  
Fin d'exécution

36

Rendre les objets d'une classe sérialisable :

```
public class Personne implements java.io.Serializable {  
    private String nom;  
    private String prenom;  
    private int    taille;  
  
    public Personne(String nom, String prenom, int taille) {  
        this.nom = nom;  
        this.taille = taille;  
        this.prenom = prenom;  
    }  
}
```

37

```
public String getNom() {  
    return this.nom;  
}  
  
public void setNom(String nom) {  
    this.nom = nom;  
}  
  
public int getTaille() {  
    return this.taille;  
}
```

38

```
public void setTaille(int taille) {  
    this.taille = taille;  
}  
  
public String getPrenom() {  
    return this.prenom;  
}  
  
public void setPrenom(String prenom) {  
    this.prenom = prenom;  
}  
}
```

39

## La sérialisation en XML

A partir de Java 1.4, le JDK permet de sérialiser un objet Java en XML plutôt que sous un format binaire en utilisant les classes **XMLEncoder** et **XMLDecoder**.

40

## La sérialisation XML propose quelques avantages :

1. une meilleure portabilité notamment dans le cas d'échanges entre JVM de différents fournisseurs
2. facilité de traitements et d'échanges sur le réseau grâce à l'utilisation du format XML

41

## Elle possède aussi plusieurs inconvénients :

1. ne peut s'utiliser par défaut que sur des objets qui respectent la convention JavaBeans (constructeur par défaut, getter/setter pour tous les attributs, ...)
2. la taille des données sérialisées est plus importante que leur équivalent binaire

42

La classe `java.beans.XMLEncoder` permet de sérialiser un objet en XML.

La classe `XMLEncoder` permet de sérialiser l'état d'un `JavaBean` dans un document XML encodé en UTF-8.

La sérialisation XML ne prend en compte que les champs pour lesquels il existe un getter et un setter public.

Le mécanisme de sérialisation XML optimise le contenu du document XML en omettant les champs dont la valeur est celle par défaut.

La classe `XMLEncoder` possède deux constructeurs :

Constructeur	Rôle
<code>XMLEncoder(OutputStream out)</code>	Créer une nouvelle instance qui utilise le flux en paramètre pour écrire le résultat de la sérialisation
<code>XMLEncoder(OutputStream out, String charset, boolean declaration, int indentation)</code>	Créer une nouvelle instance qui utilise le flux en paramètre pour écrire le résultat de la sérialisation en précisant le charset, et la taille de l'indentation.

```

XMLEncoder
public XMLEncoder(OutputStream out,
                  String charset,
                  boolean declaration,
                  int indentation)
Creates a new XML encoder to write out JavaBeans to the stream out using the given charset starting from the given indentation.
Parameters:
  out - the stream to which the XML representation of the objects will be written
  charset - the name of the requested charset, may be either a canonical name or an alias
  declaration - whether the XML declaration should be generated; set this to false when embedding the contents in another XML document
  indentation - the number of space characters to indent the entire XML document by
Throws:
  IllegalArgumentException - if out or charset is null, or if indentation is less than 0
  IllegalCharsetNameException - if charset name is illegal
  UnsupportedCharsetException - if no support for the named charset is available in this instance of the Java virtual machine
  UnsupportedOperationException - if loaded charset does not support encoding

```

```

import java.beans.XMLEncoder;
import java.io.FileOutputStream;
import java.util.Date;

public class SerializerPersonneXML {

    public static void main(String argv[]) {
        Personne personne = new Personne(new Date(), "tresBon",
"Jean", 175);
        XMLEncoder encoder = null;

```

45

```

    try {
        encoder = new XMLEncoder(new
        BufferedOutputStream(
        new FileOutputStream("personne.xml")));
        encoder.writeObject(personne);
        encoder.flush();
    } catch (java.io.IOException e) {
        e.printStackTrace();
    } finally {
        if (encoder != null) {
            encoder.close();//appelle aussi flush
        }
    } } }

```

46

## Résultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_65" class="java.beans.XMLDecoder">
  <object class="com.exemple.serialisation.Personne">
    <void property="dateNaiss">
      <object class="java.util.Date">
        <long>1367419421845</long>
      </object>
    </void>
    <void property="nom">
      <string>tresBon</string>
    </void>
    <void property="prenom">
      <string>Jean</string>
    </void>
    <void property="taille">
      <int>175</int>
    </void>
  </object>
</java>
```

47

## La classe XMLDecoder

La classe **java.beans.XMLDecoder** permet de désérialiser un objet à partir d'un document XML généré avec la classe **XMLEncoder**.

Elle possède plusieurs constructeurs :

48



Constructeur	Rôle
<code>XMLDecoder(InputStream in)</code>	Créer un nouveau décodeur pour désérialiser le document XML lu du flux en paramètre
<code>XMLDecoder(InputStream in, Object owner)</code>	Créer un nouveau décodeur pour désérialiser le document XML lu du flux en paramètre
<code>XMLDecoder(InputStream in, Object owner, ExceptionListener exceptionListener)</code>	Créer un nouveau décodeur pour désérialiser le document XML lu du flux en paramètre
<code>XMLDecoder(InputStream in, Object owner, ExceptionListener exceptionListener, ClassLoader cl)</code>	Créer un nouveau décodeur pour désérialiser le document XML lu du flux en paramètre (depuis Java 1.5)
<code>XMLDecoder(DataSource is)</code>	Créer un nouveau décodeur pour désérialiser le document XML lu du flux en paramètre (depuis Java 7)

<https://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html#XMLDecoder%28java.io.InputStream,%20java.lang.Object%29>

```
import java.beans.XMLDecoder;
import java.io.BufferedInputStream;
import java.io.FileInputStream;

public class DeserializerPersonneXML {

    public static void main(String argv[]) {
        XMLDecoder decoder = null;

        try {
            decoder = new XMLDecoder(new
                BufferedInputStream(new
                    FileInputStream("personne.xml")));
        }
    }
}
```

```
Personne personne = (Personne) decoder.readObject();
    System.out.println(personne);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (decoder != null) {
        decoder.close();
    }
}
}
```

51

## Résultat :

Personne [dateNaiss=Wed Feb 12 20:45:45 2014  
nom=tresBon, prenom=Jean, taille=175]

52

Des problèmes peuvent se poser pour les objets qui possèdent des champs eux-mêmes non sérialisables.

Dans ce cas, ces champs doivent être marqués avec le mot-clé **transient**. Cela a pour effet de les retirer du flux sérialisé.

Après désérialization, ces champs seront à null.

53

```
// création d'une classe Serializable
public class Test implements Serializable {

    // la classe String est Serializable, donc ces champs sont
    légaux
    private String nom, prenom ;

    // en revanche la classe Connection ne l'est pas,
    // il faut donc retirer ce champ de la serialization

    private transient Connection con ;
}
```

54

Pour des raisons d'optimisation, ne se trouve dans le flux sérialisé, que le minimum d'information requis pour reconstruire l'objet.

Typiquement, on y trouve donc :

- ❑ le nom complet de la classe de l'objet ;
- ❑ les noms de ses champs, et pour chacun de ces champs, son type et sa valeur.

55

Ces informations sont *a priori* suffisantes pour reconstruire l'objet.

Cela dit, il faut tout de même comprendre que la reconstruction d'un objet à partir de ces octets peut se faire dans un contexte très différent de celui de la création de ces octets.

Dans ces deux cas, il est nécessaire de vérifier que la classe que l'on possède est bien la même que celle qui a servi à la création de ces octets.

56

Vérifier son nom complet n'est pas suffisant, elle doit définir les mêmes champs, de même nom et de même type.

Pour cela, Java introduit un code de hachage associé aux classes qui implémentent `Serializable`, stocké dans un champ standard.

Ce champ standard s'appelle `serialVersionUID`, doit être de type long et doit être `private static final`.

57

Ce champ est systématiquement enregistré dans tout paquet d'octets qui représente un objet sérialisé.

S'il a été défini explicitement dans la classe de cet objet, c'est cette valeur fournie qui est utilisée.

S'il ne l'a pas été, alors la machine Java en détermine un.

58

```

public class Test implements Serializable {

    private static final long serialVersionUID = 1350092881346723535L;

    private String nom, prenom ;

    private int salaire ;

    public Test(String nom, String prenom) {
        this.nom = nom ;
        this.prenom = prenom ;
    }

    public String toString() {
        StringBuffer sb = new StringBuffer() ;
        return sb.append(nom).append(" ").append(prenom).toString() ;
    }
}

```

59

## Sérialisation :

```

File fichier = new File("tmp/fichier.txt") ;

// ouverture d'un flux sur un fichier
ObjectOutputStream ficObj = new
ObjectOutputStream(new FileOutputStream(fichier)) ;

// création d'un objet à sérializer
Test m = new Test("Marx", "Karl") ;

// sérialization de l'objet
ficObj.writeObject(m) ;

```

60

## Désérialisation :

```
File fichier = new File("tmp/fichier.txt") ;

// ouverture d'un flux sur un fichier
ObjectInputStream ficObj = new ObjectInputStream(new
FileInputStream(fichier)) ;

// désérialization de l'objet
Test m = (Test) ficObj.readObject() ;
System.out.println(m) ;
```

Beaucoup reste encore à dire.  
À vous de continuer ...

61

## Exercice 2 (XMLEncoder, XMLDecoder)

Prendre le fichiers livres.cvs qui est dans LEA.

titre;auteur;année;pages;catégorie  
Un parfait inconnu;4;1990;361;roman

Lire ce fichier et créer un ArrayList avec tous des instances de la classe Livres (attributs en rouge).

Enregistrer le contenu du ArrayList(élément par élément) en utilisant un XMLEncoder. Le fichier s'appelera livres.xml. En utilisant XMLDecoder et livres.xml afficher par System.out.println, chaque objet.

Ne pas oublier de mettre en DEPOT sous le nom  
**187NonPrénom(exercice2)**

62

## Application à Android :

### Injecter des données dans un intent

On a vu que les intents avaient un champ « extra » qui leur permet de contenir des données à véhiculer entre les applications.

Pour insérer un extra, il suffit d'utiliser la méthode **Intent.putExtra(String clé, X valeur)** avec clé la clé de l'extra et valeur la valeur associée.

type de base quelconque (int, String, double[],..)



63

Vous pouvez récupérer tous les extras d'un intent à l'aide de la méthode **Bundle.getExtras()**, auquel cas vos couples clé-valeurs sont contenus dans le Bundle.

Vous pouvez encore récupérer un extra précis à l'aide de sa clé et de son type en utilisant la méthode **getXExtra(String clé, valeurDéfaut)**, X étant le type de l'extra et valeurDéfaut la valeur qui sera retournée si la clé passée ne correspond à aucun extra de l'intent.

64



Pour les types un peu plus complexes tels que les tableaux, on ne peut préciser de valeur par défaut.

On devra utiliser la méthode `float[]`  
`getFloatArrayExtra(String key)` pour un tableau de float.

65

En règle générale, la clé de l'extra commence par le package duquel provient l'intent.

```
// On déclare une constante dans la classe maClasse
public final static String LISTE = "com.exemples.intents.LISTE"
```

```
...
```

```
Intent i = new Intent();
String[] laListe = new String[] {"Marx", "Karl"};
i.putExtra(maClasse.LISTE, laListe);
```

```
...
```

```
String[] autreListe = i.getStringArrayExtra(maClasse.LISTE);
```

66

## Les Parcelables et Sérialisables

### CAS 1 : Les Parcelables

Bundle ne peut pas prendre tous les objets, il faut qu'ils soient sérialisables.

Or, dans le cas d'Android, on considère qu'un objet est sérialisable à partir du moment où il implémente correctement l'interface Parcelable.

Si on devait entrer dans les détails, sachez qu'un Parcelable est un objet qui sera transmis à un Parcel, et que l'objectif des Parcel est de transmettre des messages entre différents processus du système.

Utiliser les hyperliens pour avoir les détails.

67

Pour implémenter l'interface Parcelable, il faut redéfinir deux méthodes :

- ❑ **int describeContents()**, qui permet de définir si vous avez des paramètres spéciaux dans votre Parcelable.

En ce moment les seuls objets spéciaux à considérer sont les FileDescriptor. Ainsi, si votre objet ne contient pas d'objet de type FileDescriptor, vous pouvez renvoyer 0, sinon renvoyez **Parcelable.CONTENT\_FILE\_DESCRIPTOR**.

68

- ❑ `void writeToParcel(Parcel dest, int flags)`, avec `dest` le `Parcel` dans lequel nous allons insérer les attributs de notre objet et `flags` un entier qui vaut la plupart du temps `0`.

C'est dans cette classe que nous allons écrire dans le `Parcel` qui transmettra le message.

### Doc Android :

`dest`

The Parcel in which the object should be written.

`flags`

Additional flags about how the object should be written. May be 0 or `PARCELABLE_WRITE_RETURN_VALUE`.

69

Nota : Les attributs sont à insérer dans le `Parcel` dans l'ordre dans lequel ils sont déclarés dans la classe !

```
import android.os.Parcel;
import android.os.Parcelable;
```

```
public class Contact implements Parcelable{
    private String Nom;
    private String Prenom;
    private int Numero;

    public Contact(String Nom, String Prenom, int Numero) {
        this.Nom = Nom;
        this.Prenom = Prenom;
        this.Numero = Numero; }
}
```

70

```

@Override
public int describeContents() {
    //On renvoie 0, car notre classe ne contient pas de FileDescriptor
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    // On ajoute les objets dans l'ordre dans lequel on les a déclarés
    dest.writeString(Nom);
    dest.writeString(Prenom);
    dest.writeInt(Numero);
}
}

```

71

Tous nos attributs sont désormais dans le **Parcel**, on peut transmettre notre objet.

Il nous faut encore ajouter un champ statique de type **Parcelable.Creator** et qui s'appellera impérativement « **CREATOR** », sinon nous serions incapables de reconstruire un objet qui est passé par un Parcel.

72

```

public static final Parcelable.Creator<Contact> CREATOR =
new Parcelable.Creator<Contact>() {
    @Override
    public Contact createFromParcel(Parcel source) {
        return new Contact(source);
    }
    @Override
    public Contact[] newArray(int size) {
        return new Contact[size];
    }
}; //autre constructeur
public Contact(Parcel in) {
    Nom = in.readString();
    Prenom = in.readString();
    Numero = in.readInt();
}

```

Crée un tableau initialisé à null

Crée une nouvelle instance de la classe Contact dont les données sont prises de **source** qui a été créée préalablement par **writeToParcel(Parcel dest, int flags)**

73

## De la doc de Android

### Public Methods

`public abstract T createFromParcel (Parcelable source)`

Create a new instance of the Parcelable class, instantiating it from the given Parcel whose data had previously been written by `Parcelable.writeToParcel()`.

#### Parameters

*source* The Parcel to read the object's data from.

#### Returns

Returns a new instance of the Parcelable class.

`public abstract T[] newArray (int size)`

Create a new array of the Parcelable class.

#### Parameters

*size* Size of the array.

#### Returns

Returns an array of the Parcelable class, with every entry initialized to null.

74

Comme n'importe quel autre objet, on peut l'ajouter dans un intent avec `putExtra` et on peut le récupérer avec `getParcelableExtra`.

```
Intent i = new Intent();
Contact c = new Contact("Marx", "Karl", 06);
i.putExtra("com.examples.intent.CONTACT", c);

.....

Contact c1 = i.getParcelableExtra
("com.examples.intent.CONTACT", );
```

75

On peut passer des conteneurs :

//Passage d'un ArrayList d'objets parcelables

//Exemple pour une liste de contacts

```
ArrayList<Contact> listeContacts;
```

```
Intent monInt= new Intent(DeActivity.this,
VersActivity.class);
monInt.putParcelableArrayListExtra("clé unique",
listeContacts);
startActivity(monInt);
```

76

//Obtenir listeContacts dans VersActivity

```
Intent leInt = getIntent();
```

```
ArrayList<Contact> listeContacts =  
leInt.getParcelableArrayList("clé unique");
```

77

## CAS 2 : Les Sérialisables

### Exemple d'une classe sérialisable :

Vous avez le projet de tout le code dans LEA

Projet : ParcelableSerialisable

```
package com.example.parcelableserialisable;  
import java.io.Serializable;  
public class Personne implements Serializable {  
    private static final long serialVersionUID = -  
7060210544600464481L;  
    private String nom;  
    private int age;  
  
    public String getNom() {  
        return nom;    }  
}
```

78

```

public void setNom(String nom) {
    this.nom = nom;    }

public int getAge() {
    return age;
    }
    public void setAge(int age) {
    this.age = age;
    }
}

```

79

## Envoi d'un objet sérialisé :

```

public void MethodeSerialise(){
    Personne laPersonne = new Personne();
    laPersonne.setNom("Antonio");
    laPersonne.setAge(25);
    Intent intSer = new
Intent(this,MontrerSerialisableActivity.class);
    Bundle donnees = new Bundle();
    donnees.putSerializable(SER_KEY,laPersonne);
    intSer.putExtras(donnees);
    startActivity(intSer);
}

```

80



### Récupérer les données sérialisées dans une autre activité :

```
TextView infos = new TextView(this);
```

```
Personne laPersonne =  
(Personne) getIntent().getSerializableExtra(ParseSeriaActivi  
ty.SER_KEY);
```

```
infos.setText("Votre nom : " + laPersonne.getNom() + "\n"+  
"Votre âge : " + laPersonne.getAge());
```

```
setContentView(infos);
```

81

### Exemple d'une classe parcelable :

```
import android.os.Parcel;  
import android.os.Parcelable;  
public class Film implements Parcelable {  
    private String num;  
    private String titre;  
    private int duree;  
  
    public String getNum() {  
return num;  
    }  
    public void setNum(String num) {  
this.num = num;  
    }  
}
```

82

```

public String getTitre() {
    return titre;
}

public void setTitre(String titre) {
    this.titre = titre;
}

public int getDuree() {
    return duree;
}

public void setDuree(int duree) {
    this.duree = duree;
}

```

83

```

public static final Parcelable.Creator<Film>
CREATOR = new Creator<Film>() {
    public Film createFromParcel(Parcel source) {
        Film leFilm = new Film();
        leFilm.num = source.readString();
        leFilm.titre = source.readString();
        leFilm.duree = source.readInt();
        return leFilm;
    }
    public Film[] newArray(int size) {
        return new Film[size];
    }
};

```

84

```

public int describeContents() {
return 0;
}
public void writeToParcel(Parcel parcel, int flags) {
    parcel.writeString(num);
    parcel.writeString(titre);
    parcel.writeInt(duree);
}
}

```

85

## Envoi d'un objet «parcelé»

```

public void MethodeParcelable(){
    Film leFilm = new Film();
    leFilm.setNum("12345");
    leFilm.setTitre("Antonio le Grand");
    leFilm.setDuree(75);
    Intent intPar = new
Intent(this,MontrerParcelableActivity.class);
    Bundle donnees = new Bundle();
    donnees.putParcelable(PAR_KEY, leFilm);
    intPar.putExtras(donnees);

    startActivity(intPar);
}

```

86

## Récupérer les données «parcelées» dans une autre activité :

```
TextView infos = new TextView(this);
```

```
Film leFilm =  
(Film) getIntent().getParcelableExtra(ParseSeriaActivity.PA  
R_KEY);
```

```
infos.setText("Numéro du film : " + leFilm.getNum()+"\n"+  
    "Titre : " + leFilm.getTitre() + "\n" +  
    "Durée : " + leFilm.getDuree());  
setContentView(infos);
```

87

## Conclusion

1. Parcelable est plus rapide que l'interface Serializable
2. Interface parcelable prend plus de temps pour l'implémentation par rapport à l'interface Serializable
3. Interface Serializable est plus facile à mettre en œuvre
4. Interface Serializable créer un grand nombre d'objets temporaires et cause des appels au «garbage collector»
5. Tableau parcelable peut être passé par des intents dans android

88

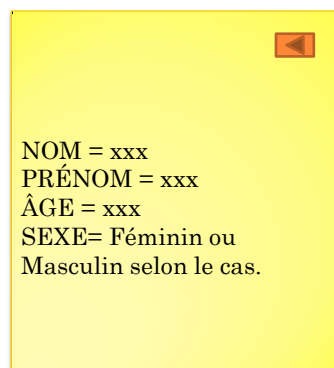
### Exercice 3:

Créer une classe Personne «Parcelable». Dans l'activité 1 on va créer un `ArrayList<Personne> listePersonne`. Vous y allez placer quelques personnes. On envoie `listePersonnes` à l'activité 2 que les affiche. Prévoir un bouton de retour à l'activité principale.

Activité 1



Activité 2



89

### Intents implicites

#### Projet : Intent\_Implicite

On envoie une requête à un destinataire, sans savoir qui il est, et d'ailleurs cela n'a pas d'importance tant que le travail qu'on lui demande de faire est effectué.

Ainsi, les applications destinataires sont soit fournies par Android, soit par d'autres applications téléchargées sur Google Play par exemple.

90

## Les données

### L'URI

Un URI est une chaîne de caractères qui permet d'identifier un endroit. Sont très similaires à un URL.

Un peu à la manière d'un serveur, nos fournisseurs de contenu vont répondre en fonction de l'URI fournie.

91

## Syntaxe d'un URI

**<schéma> : <information> { ? <requête> } { # <fragment> }**

Le schéma décrit quelle est la nature de l'information.

**tel** pour téléphone

**http** site internet

**autres**

tel :5143895921  
geo:123.456789,-12.345678  
content://contacts/people/

Permet de fournir une précision par rapport à l'information.

Permet d'accéder à une sous-partie de l'information.

92


Pour créer un objet URI, il suffit d'utiliser la méthode statique `Uri.parse(String uri)`. Par exemple, pour envoyer un SMS à une personne

```
Uri sms = Uri.parse("sms:123456789");
```

Mais on peut aussi indiquer plusieurs destinataires et un corps pour ce message :

```
Uri sms =  
Uri.parse("sms:12345678,87654321?body=Salut%20les%20p  
otes");
```

Contenu de la chaîne doit être encodé.



93

## L'action

Une action est une constante qui se trouve dans la classe Intent et qui a la forme ACTION\_XXX, où XXX peut être :

VIEW – pour voir quelque chose

Par exemple :

```
intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.google.com"));
```

94

Pris du projet `Intent_Implicite` qui vous avez dans le dossier Partage.

```
intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.google.com"));
```

```
intent = new Intent(Intent.ACTION_CALL,  
Uri.parse("tel:(+514)3895921"));
```

```
intent = new Intent(Intent.ACTION_DIAL,  
Uri.parse("tel:"));  
startActivity(intent);
```

95

```
intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("geo:50.123,7.1434?z=19"));
```

```
intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("geo:0,0?q=query"));
```

```
intent = new  
Intent("android.media.action.IMAGE_CAPTU  
RE");
```

96



```
intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("content://contacts/people/"));
```

```
intent = new Intent(Intent.ACTION_EDIT,  
Uri.parse("content://contacts/people/1"));
```

97

## Site Android :

<http://developer.android.com/reference/android/content/Intent.html>

### Quelques actions standards

[ACTION\\_MAIN](#)

[ACTION\\_VIEW](#)

[ACTION\\_EDIT](#)

[ACTION\\_PICK](#)

[ACTION\\_CHOOSER](#)

[ACTION\\_GET\\_CONTENT](#)

[ACTION\\_DIAL](#)

[ACTION\\_CALL](#)

[ACTION\\_SEND](#)

[ACTION\\_SENDTO](#)

[ACTION\\_ANSWER](#)

[ACTION\\_INSERT](#)

[ACTION\\_DELETE](#)

98

## Cas d'utilisation

```
monBouton.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v) {
        Uri telephone = Uri.parse("tel:123456789");
        Intent secondeActivite = new
Intent(Intent.ACTION_DIAL, telephone);
        startActivity(secondeActivite);
    }
});
```

99

## Dans le Manifest

```
<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="17" />

<uses-permission
    android:name="android.permission.CALL_PHONE" >
</uses-permission>
<uses-permission
    android:name="android.permission.CAMERA" >
</uses-permission>
<uses-permission
    android:name="android.permission.READ_CONTACTS" >
</uses-permission>
```

100

```
<uses-permission
  android:name="android.permission.INTERNET" />
</uses-permission>
```

```
<application
  android:allowBackup="true"
```

```
.....
```

101

Vous pouvez vérifier si un composant va réagir à un intent à l'aide de [Package Manager](#).

Le Package Manager est un objet qui vous permet d'obtenir des informations sur les packages qui sont installés sur l'appareil.

```
Intent leInt = new Intent(Intent.ACTION_VIEW,
  Uri.parse("tel:123456789"));
```

```
PackageManager manager = getPackageManager();
```

```
ComponentName component =
  leInt.resolveActivity(manager);
if(component != null)
```

```
//Alors c'est qu'il y a une activité qui va gérer l'intent
```

102

Beaucoup encore à découvrir.

À vous de jouer ...

103

Voyella:

`de.vogella.android.intent.implicit`

<http://www.vogella.com/code/de.vogella.android.intent.explicit/codestartpage.html>

<http://www.vogella.com/code/de.vogella.android.intent.implicit/codestartpage.html>

<http://www.vogella.com/tutorials/Git/article.html>

104

Intents.pdf

<http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/les-intents-explicites>

<http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/la-communication-entre-composants>

Sérialisation :

<http://www.jmdoudoux.fr/java/dej/chap-serialisation.htm>

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialisation.html>

105