Non-bloquant – les canaux (Channel)

* La première interface importante est l'interface Channel de la librairie **java.nio.channels**
* La plupart des classes que nous utiliserons pour nos entrées/sorties seront des Channel
* Cette interface ne contient que de deux méthodes
  + **boolean isOpen()**
  + **void close() throws IOException**
* Ce qui va nous intéresser, ce sont les déclinaisons de cette classe, en particulier
  + **SocketChannel**, **DatagramChannel** et **ServerSocketChannel**
* Ces classes vont substituées les classes :
  + **Socket**, **DatagramSocket** et **ServerSocket**

PR - E/S NON BLOQUANT 11

Les canaux (Channel)

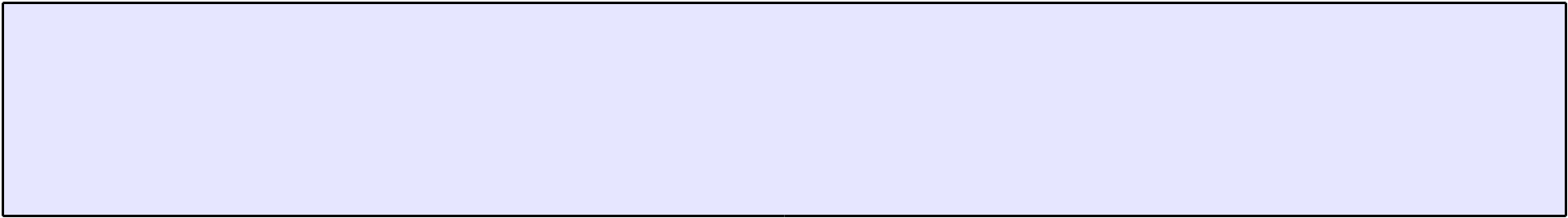
* Pour les trois classes de canaux qui nous intéressent : **SocketChannel**, **DatagramChannel** et **ServerSocketChannel**
  + Il n'y aura pas de constructeur
  + Pour obtenir un nouveau canal, on utilisera la méthode statique **open()** des classes correspondantes
  + De plus pour ces classes, on peut dire si on souhaite que les objets soient bloquant ou non
    - on utilise la méthode **SelectableChannel configureBlocking(boolean block)**
  + On peut aussi tester si un canal est bloquant ou non
    - **boolean isBlocking()**

PR - E/S NON BLOQUANT 12

SocketChannel

* La classe **SocketChannel** va correspondre à une Socket en TCP
* On peut l'utiliser pour ce connecter à une machine
  + **public abstract boolean connect(SocketAddress r) throws IOException**
  + Attention si la **SocketChannel** est déclarée non-bloquante, on peut sortir du connect sans être connecté (la valeur renvoyée est false)
  + La méthode **boolean finishConnect()** permet de tester si la connexton est finie
  + Pour contourner ce problème, on peut faire :

**sc.configureBlocking(false);**



**sc.connect(new InetSocketAddress("localhost", 5555)); while(! sc.finishConnect() ){**

**//wait, or do something else...**

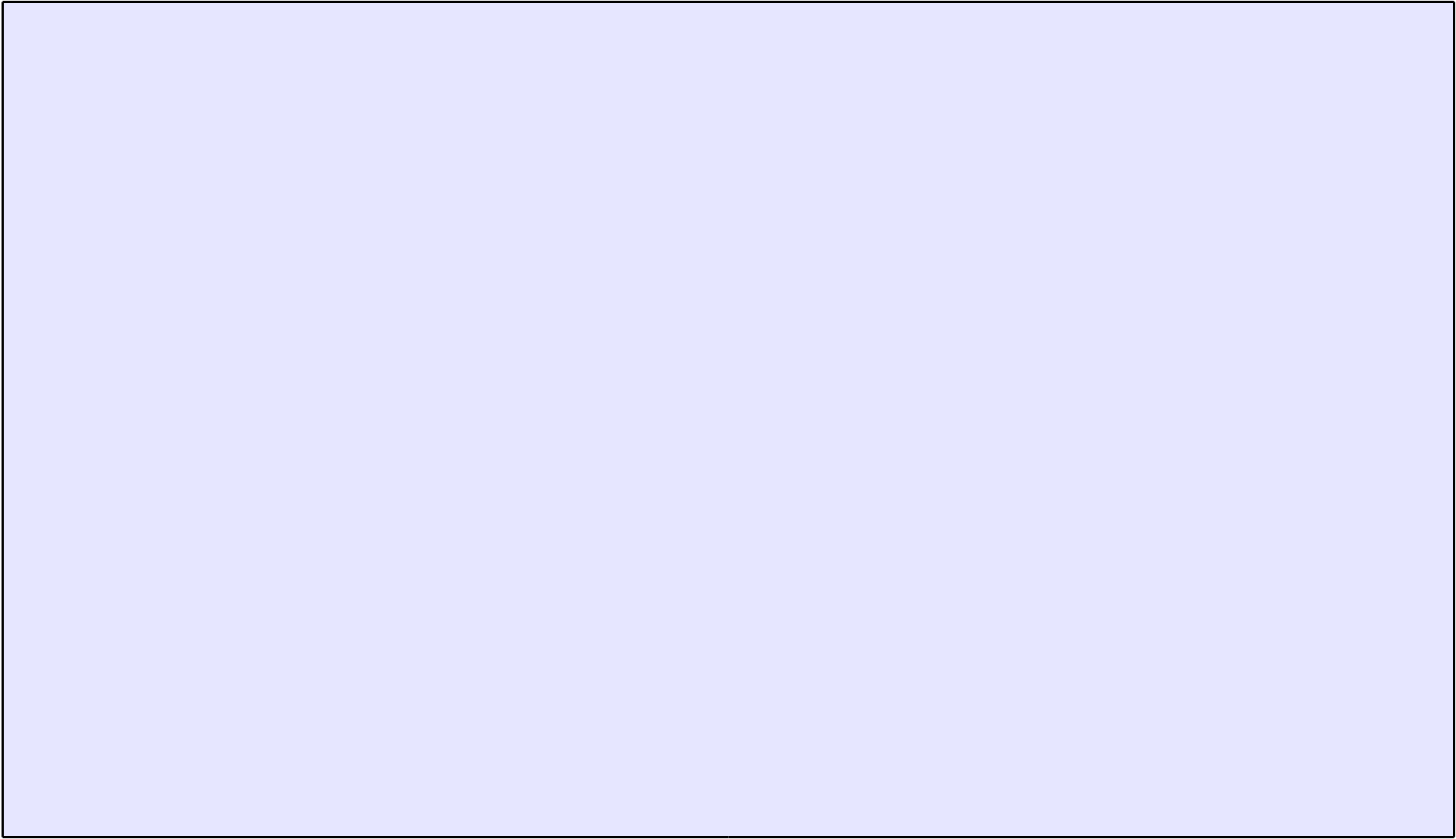
**}**

●

* On peut l'utiliser pour lire et écrire des données
  + **abstract int read(ByteBuffer dst)**
  + **abstract int write(ByteBuffer src)**
* On peut récupérer la **Socket** associée avec la méthode **socket()**

PR - E/S NON BLOQUANT 13

Exemple

**public class DoubleRead2{**

**public static void main(String[] args){**

**try{**

**Selector sel=Selector.open();**

**DatagramChannel dsc1=DatagramChannel.open(); DatagramChannel dsc2=DatagramChannel.open(); dsc1.configureBlocking(false); dsc2.configureBlocking(false);**

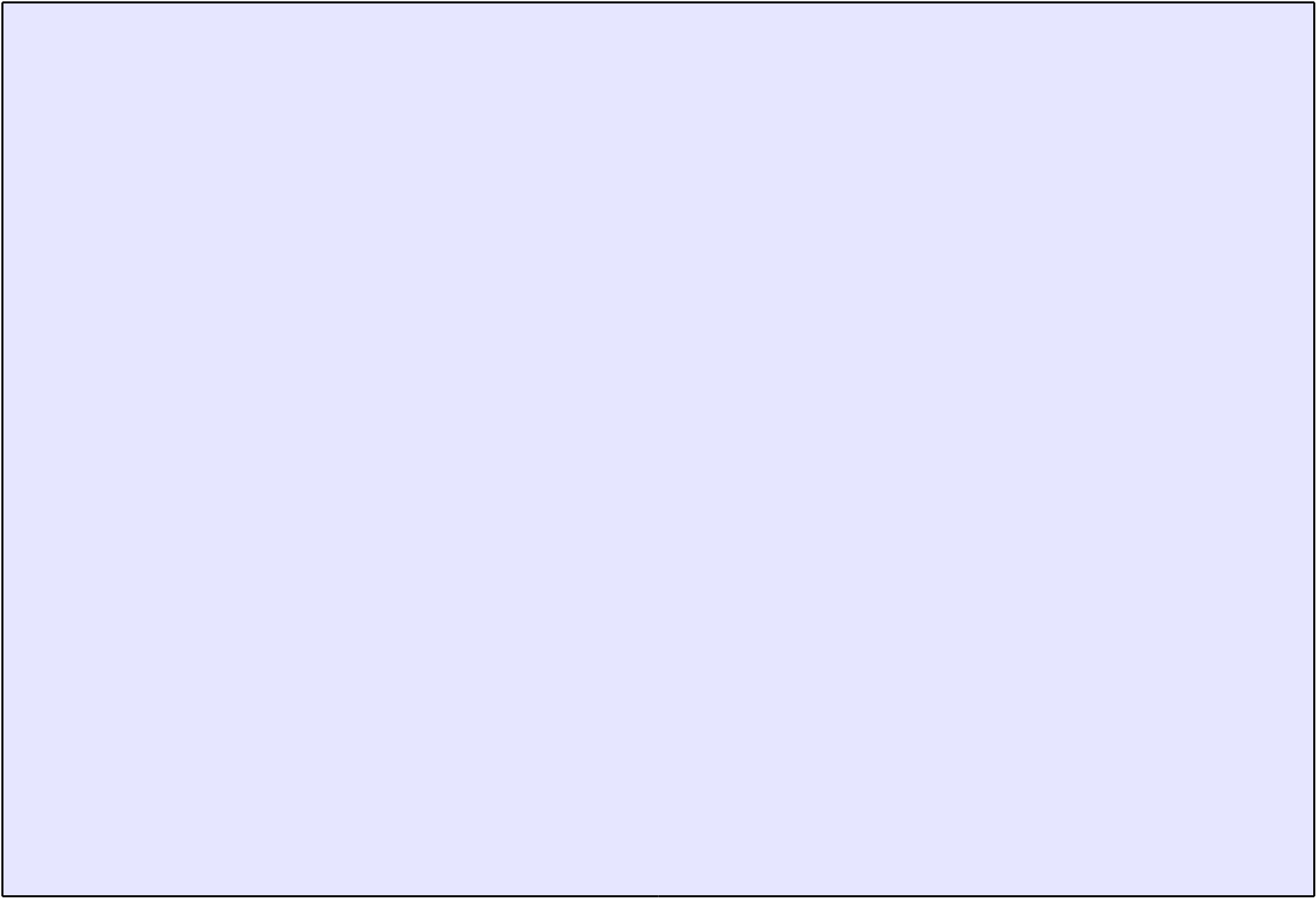
**dsc1.bind(new InetSocketAddress(4344)); dsc2.bind(new InetSocketAddress(4343));**

**dsc1.register(sel,SelectionKey.OP\_READ); dsc2.register(sel,SelectionKey.OP\_READ);**

PR - E/S NON BLOQUANT 22

**while(true){**

Exemple (suite)



**System.out.println("Waiting for messages"); sel.select();**

**Iterator<SelectionKey> it=sel.selectedKeys().iterator(); while(it.hasNext()){**

**SelectionKey sk=it.next();**

**it.remove();**

**if(sk.isReadable() && sk.channel()==dsc1){**

**ByteBuffer buff=ByteBuffer.allocate(100); System.out.println("Message UDP 4344 recu"); dsc1.receive(buff);**

**String st=new String(buff.array(),0,buff.array().length); buff.clear();**

**System.out.println("Message :"+st);**

**} else if (sk.isReadable() && sk.channel()==dsc2){**

**ByteBuffer buff=ByteBuffer. allocate(100); System.out.println("Message UDP 4343 recu"); dsc2.receive(buff);**

**String st=new String(buff.array(),0,buff.array().length); System.out.println("Message :"+st);**

**} else{**

**System.out.println("Que s'est il passe");**

**}**

**}**

**}**

**} catch(Exception e){**

**e.printStackTrace();}**