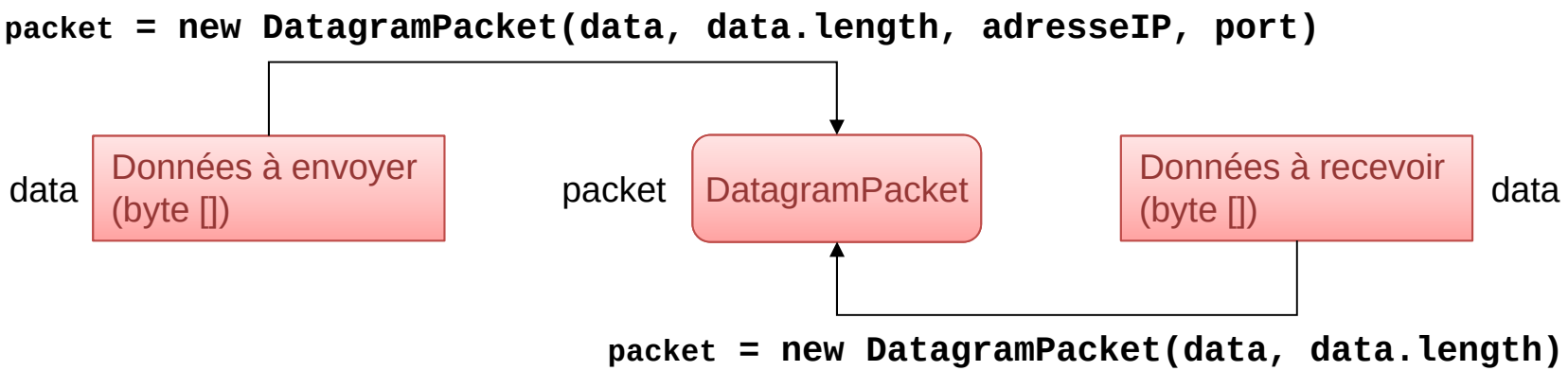
Sockets en mode non connecté

- Deux classes
  - **`DatagramSocket`** : un seul type de socket
  - **`DatagramPacket`** : format de message pour UDP
    - Conversion entre données et paquets (dans les 2 sens)
    - Les DatagramPacket sont construits différemment pour l'envoi et la réception



Classe java.net.DatagramSocket

- Cette classe permet d'envoyer et de recevoir des paquets (UDP)
- Constructeurs
  - `DatagramSocket()`
  - `DatagramSocket(int port)`
  - Construit une socket datagramme en spécifiant éventuellement un port sur la machine locale (par défaut, un port disponible quelconque est choisi)
  - Levée éventuelle de `SocketException`

Classe java.net.DatagramSocket

- Emission
  - `void send(DatagramPacket p)`
    - Levée éventuelle de `IOException`
- Réception
  - `void receive(DatagramPacket p)`
- Ces opérations permettent d'envoyer et de recevoir un paquet
  - Un paquet est un objet de la classe `java.net.DatagramPacket` qui possède une zone de données et (éventuellement) une adresse IP et un numéro de port (destinataire dans le cas *send*, émetteur dans le cas *receive*)

Classe java.net.DatagramSocket

- `void close()` : fermeture de la socket ; penser à toujours fermer les sockets qui ne sont plus utilisées
- `int getLocalPort()` : retourne le port sur lequel on écoute, ou le port anonyme sur lequel on a envoyé le paquet
- Il est possible de « connecter » une socket datagramme à un destinataire
  - `void connect(InetAddress ia, int p)`
  - Dans ce cas, les paquets émis sur la socket seront toujours pour l'adresse spécifiée
  - La connexion simplifie l'envoi d'une série de paquets (il n'est plus nécessaire de spécifier l'adresse de destination pour chacun d'entre eux) et accélère les contrôles de sécurité (ils ont lieu une fois pour toute à la connexion)
  - La « déconnexion » avec `void disConnect()` défait l'association (la socket redevient disponible comme dans l'état initial)
  - `InetAddress getInetAddress()` : retourne l'adresse à laquelle on est connecté
  - `int getPort()` : retourne le port auquel on est connecté

Classe java.net.DatagramPacket

- Emission
  - `DatagramPacket (byte[] buf, int length, InetAddress ia, int port)`
    - construit un datagramme destiné à être envoyé à l'adresse `ia`, sur le port `port`
    - le tampon `buf` de longueur `length` contient les octets à envoyer
- Réception
  - `DatagramPacket (byte[] buf, int length)`
    - construit un datagramme destiné à recevoir des données
    - le tampon `buf` de longueur `length` contient les octets à recevoir

Classe java.net.DatagramPacket

- `InetAddress getAddress()` : retourne l'adresse de l'hôte d'où vient ou où va le paquet
- `void setAddress(InetAddress ia)` : change l'adresse où envoyer le paquet
- `int getPort()` : le port de l'hôte qui a envoyé, ou vers lequel on envoie
- `void setPort(int port)` : change le port vers lequel on envoie le paquet
- `byte[] getData()` : retourne le tableau contenant les données à envoyer ou reçues
- `void setData(byte[] d)` : change le tableau des données à envoyer
- `int getLength()` : retourne la taille du tableau des données reçues
- `void setLength(int length)` : change la taille du tableau des données à envoyer

Exemple

Pour l'envoi de datagramme

```
try{
   byte [] tampon;
   tampon = . . . ;
   InetAddress ia = . . .;
   int port = . . .;
   DatagramPacket envoiPaquet = new DatagramPacket (
                tampon, tampon.length, ia, port);
   DatagramSocket socket = new DatagramSocket();
   socket.send(envoiPaquet);
}catch(SocketException se){
}catch(IOException ioe){
}
```

Pour la réception de datagramme

```
try{
   byte [] tampon = new byte[];
   int port = . . .;
   DatagramPacket receptionPaquet = new DatagramPacket (
                                tampon, tampon.length);
   DatagramSocket socket = new DatagramSocket(port);
   socket.receive(receptionPaquet);
   String s = new String (receptionPaquet.getData());
}catch(SocketException se){
}catch(IOException ioe){
}
```

Client / serveur en mode non connecté

Sur un client

envoi requête

```
DatagramSocket socket = new DatagramSocket();

byte [] envoiTampon = new byte[1024];
byte [] receptionTampon = new byte[1024];
String requete, reponse;

requete = … // dépend de l'application
envoiTampon = requete.getBytes();
DatagramPacket envoiPaquet = new DatagramPacket(
    envoiTampon, envoiTampon.length,
    InetAddress.getbyname("prevert.upmf-grenoble.fr"), 8254);
socket.send(envoiPaquet);

DatagramPacket receptionPaquet = new DatagramPacket(
    receptionTampon, receptionTampon.length);
socket.receive(receptionPaquet);
reponse = new String(receptionPaquet.getData());
```

doit être connu du client

réception réponse

Sur le serveur (prevert.upmf-grenoble.fr)

réception requête

```
DatagramSocket socket = new DatagramSocket(8254);

byte [] receptionTampon = new byte[1024];
byte [] envoiTampon = new byte[1024];
String requete, reponse;

while (true) {
    DatagramPacket receptionPaquet = new DatagramPacket(
        receptionTampon, receptionTampon.length);
    socket.receive(receptionPaquet);
    requete = new String(receptionPaquet.getData());

    // déterminer adresse et port du client
    InetAddress clientAdresse = receptionPaquet.getAddress();
    int clientPort = receptionPaquet.getPort();
    // exécuter le service
    …
    // envoyer la réponse au client
    envoiTampon = reponse.getBytes();
    DatagramPacket envoiPaquet = new DatagramPacket(
        envoiTampon, envoiTampon.length, clientAdresse,
        clientPort);
    socket.send(envoiPaquet);
}
```

envoi réponse