java.net.URI

- Depuis j2sdk1.4: manipulation syntaxique des URI
  - Neuf composants répertoriés sur un URI
    - **scheme** String
    - **scheme-specific-part** String
    - **authority** String
    - **user-info** String
    - **host** String
    - **port** int
    - **path** String
    - **query** String
    - **fragment** String

Classe URI

- Constructeur acceptant la chaîne URI en argument
  - Vérifie la syntaxe des URI (RFC 2396):
  - URI u1 = new URI("http://igm.univ-mlv.fr");
  - URI u2 = new URI("nimporte:quoi");
  - URI u3 = new URI("relatif");
  - URI u4 = new URI("\\\interdit");
    // lève une **URISyntaxException**
- Méthode statique create(String uri) identique
  - Mais lève une **IllegalArgumentException** qu'on a pas besoin de récupérer ni de propager (RuntimeException).

Classe URI (suite)

- Quatre autres constructeurs spécifiant des parties
  - **URI(String schema, String user, String machine, int port, String path, String query, String fragment)**
    Ex:URI("ftp", "user", "www.test.fr", 21, "/d/file.txt", "param=val", "ancre") qui correspond à ftp://user@www.test.fr:21/d/file.txt?param=val#ancre
  - **URI(String schema, String autorité, String path, String query, String fragment)**
    Ex:URI("ftp", "user@www.test.fr:21", "/d/file.txt", "param=val", "ancre")
  - **URI(String schema, String partieDépendante, String fragment)**
    Ex:URI("ftp", "user@www.test.fr:21/d/file.txt?param=val", "ancre")
  - **URI(String schema, String machine, String path, String fragment)**
    équivalent à un appel au premier constructeur avec
    URI(schéma, null, machine, -1, path, fragment)
  - Plus généralement, null ou -1 pour un champ absent

Classe URI (spécificités)

- Principal avantage: permet d'utiliser des caractères Unicode en plus des caractères ASCII
  - **URI ipv6 = new URI("sch","1080::02C:417A","/mon fichier",null);**
    System.out.println(ipv6);
    // Affiche: sch:[1080::02C:417A]/mon%20fichier
  - « Analyse » des parties de l'URL, w.r.t le constructeur
    URI bad1 = new URI ("sch://500.1.2.3/");
    System.out.println(bad1); // Affiche: sch://500.1.2.3/
    l'authorité de nommage pourrait être considérée comme non liée à un serveur
  - **URI bad2 = new URI ("sch", "500.1.2.3", "/", null);**
    System.out.println(bad2);
    // lève: URISyntaxException: Malformed IPv4 address...
    ici, le 2nd argument est (syntaxiquement) une adresse IP

Classe URI (accesseurs)

- Découpage d'un objet URI en « morceaux »
  - getScheme(), getUserInfo(), getHost(), getAuthority(), getPort(), getPath(), getQuery(), getFragment()
  - isOpaque(), getSchemeSpecificPart()
  - Méthode parseServerAuthority() vérifie que l'autorité correspond à des informations relatives à un serveur
    - Sinon lève **URISyntaxException**
  - get**Raw**UserInfo(), get**Raw**Authority(), get**Raw**Path(), etc... donnent les champs avant décodage Unicode
  - getASCIIString() donne cet URI en US-ASCII

Classe URI (relatifs et absolus)

- Dans le cas d'un URI représentant un URI relatif, toutes les méthodes correspondant à une partie manquante retournent null
- Méthode **resolve(URI relatif)**
  - Retourne l'URI absolu correspondant à l'URI realtif passé en argument, résolu par rapport à l'URI de base sur lequel la méthode est appelée
  - Existe avec un argument de type **String**
- Méthode **relativize(URI uri)**
  - Retourne l'URI relatif qui représente **uri** par rapport à l'URI de base sur lequel est appelé la méthode
- Méthode **normalize()** élimine les parties inutiles
  - http://machine/dir/./subdir/../file.html => http://machine/dir/file.html

Classes URI et URL

- Instance de la classe URI
  - Référence à la RFC 2396, au sens syntaxique du terme.
  - Peut être absolu ou relatif
  - Analysé selon la syntaxe générique, indépendamment du schéma.
  - Pas de résolution du nom de l'hôte, ni gestionnaire de schéma.
  - Égalité, hachage et comparaison sur les caractères décrivant l'URI.
- Instance de la classe URL
  - Composants syntaxiques + informations pour l'accès à la ressource.
  - Doit être absolu, c'est-à dire spécifier un schéma.
  - Analysé en fonction du schéma.
  - Donne lieu à l'établissement d'un gestionnaire de schéma associé.
  - Ne peut pas créer d'URL pour un schéma dont on a pas le gestionnaire.
  - Égalité, hachage et comparaison dépendent du schéma et de l'hôte.

Classe URL

- Historiquement antérieure à URI
  - Ne respecte pas rigoureusement la terminologie des URI (RFC 2396)
  - Interprétation restricitve (ex: schéma = protocole, path = file)
  - **Aucun « codage »** sur les morceaux de l'URL
- Comment contruire un URL (**MalformedURLException**)
  - **URL url = new URI(...).toURL();**
  - Six constructeurs
    - **URL(String spec)**
    - **URL(String scheme, String host, int port, String path)**
    - **URL(String scheme, String host, String path)**
    - **URL(URL context, String spec)**
    - **URL(URL context, String spec, URLStreamHandler)**
    - **URL(String scheme, String host, int port, String path, URLStreamHandler)**

URL: accès aux caractéristiques

- Méthodes d'accès aux différentes parties de l'URL
  - getProtocol(), getAuthority(), getHost(), getPort(), getDefaultPort(), getFile() (avec <query>), getPath() (sans <query>), getQuery(), getRef()
  - toExternalForm() retourne la forme canonique de l'URL
- Deux méthodes de comparaison
  - equals() retourne true si les 2 URL sont différents de null, et référencent exactement la même ancre de la même ressource sur la même machine
  - sameFile() identique mais retourne true même si les ancres diffèrent

Comparaison entre URIs / entre URLs

- La classe URI ne s'occupe « que » des caractères

```
URI uri1 = new URI("http://igm.univ-mlv.fr/~duris/index.html");
URI uri2 = new URI("http://monge.univ-mlv.fr/~duris/index.html");
System.out.println(uri1.equals(uri2));      // Affiche: false
```

- La classe URL « analyse » la localisation du contenu

  ...plus particulièrement la machine et le port

```
URL url1 = new URL("http://igm.univ-mlv.fr/~duris/index.html");
URL url2 = new URL("http://monge.univ-mlv.fr/~duris/index.html");
URL url3 = new URL("http://193.55.63.80:80/~duris/index.html");
System.out.println(url1.equals(url2));      // Affiche: true
System.out.println(url1.equals(url3));      // Affiche: true
```


URL: accès au contenu

- La méthode getContent() retourne un objet correspondant à la ressource
  - URL u = new URL("http://igm.univ-mlv.fr/~duris/logo.gif");
    System.out.println(u.getContent());
    // Affiche: sun.awt.image.URLImageSource@a0dcd9
- Récupérer un flot de lecture sur cette ressource
  - InputStream openStream()
    - Pas toujours très pertinent
- Ces deux méthodes sont des raccourcis de
  - openConnection().getContent()
  - openConnection().getInputStream()

Classe URLConnection

- **Gestionnaire de connection**
- Classe abstraite commune à tous les objets qui gèrent les connexions vers des ressources référencées par des URL
  - Méthodes « indépendantes » du schéma
  - Récupération d'attributs de la connexion
  - Passage d'attributs à la connexion
- En général, une instance de **URLConnection** est créée par un appel à **openConnection()** sur un objet **URL**

Cycle de vie de URLConnection

- 1. L'objet représentant la connexion est créé par invocation de la méthode openConnection() sur un URL donné
- 2. Les paramètres de cette (future) connexion peuvent être manipulés
- 3. La connexion est réellement établie par un appel à la méthode **connect()**
- 4. L'objet (la ressource) distant(e) peut être accédé(e) ainsi que les champs d'en-tête associés à cette connexion

Manipuler les paramètres de connexion

- Les paramètres de la connexion doivent être fixés avant qu'elle soit effectivement établie
  - setRequestProperty(String key, String value)
  - getRequestProperty(String key)
  - Méthodes dédiées à certains paramètres: doInput, doOutput, allowUserInteraction, ifModifiedSince, useCaches...
  - Les méthodes setDefaultAllowUserInteraction() et setDefaultUseCaches() donnent des valeurs par défaut au paramètre correspondant pour toutes les futures connexions

Paramètres de connexion: exemple

- ```
  // Création de l'url
  URL url = new URL("http://www.univ-mlv.fr/index.html");
  // Création de l'objet connexion
  URLConnection uc = url.openConnection();
  // spécification de paramètres pour la connexion
  uc.setRequestProperty("Connection","close");
  SimpleDateFormat f = new SimpleDateFormat("dd/MM/yy zzzz");
  Date d = f.parse("01/01/2002 GMT", new ParsePosition(0));
  uc.setIfModifiedSince(d.getTime());
  ```
- Génère les en-têtes suivants dans la requête
  Connection: close
  If-Modified-Since: Tue, 01, Jan 2002 00:00:00 GMT
- Et d'autres, par défaut, dont
  User-Agent: Java/1.4.2_03

Utiliser la connexion

- Commencer par appeler connect()
  - La plupart des méthodes suivantes font implicitement un appel à connect() s'il n'a pas déjà été fait
- Exploitation de la connexion
  - getContent() donne la ressource sous la forme d'objet
  - getHeaderField() donne la valeur d'un champ d'en-tête
  - getInputStream() flot de lecture sur le contenu
  - getOutputStream() flot d'écriture sur la connexion
  ```

Récupération des champs d'en-tête

- getHeaderField() est décliné pour certains champs d'en-tête spécifiques (valeurs de retour de type String ou int):
  - getContentType(), getContentEncoding(), getContentLength(), getDate(), getExpiration(), getLastModified()
- Méthodes utilitaires
  - int getHeaderFieldInt(String key, int defaut)
    long getHeaderFieldDate(String key, long defaut)
    String getHeaderFieldKey(int n)
    String getHeaderField(int n)

Sous-classes de URLConnection

- **HttpURLConnection** (classe abstraite)
  - Méthodes d'accès spécifiques au protocole HTTP, ex:
  - setRequestMethod() spécifie la méthode de requête
  - getResponseCode() et getResponseMessage() donnent le code et le message de réponse (constantes définies)
  - setFollowRedirects() pour suivre les redirections (3xx) pour les instances de cette classe de gestionnaire, et setInstanceFollowRedirects() pour une instance particulière
  - disconnect() demande la déconnexion si *keep alive*
  - usingProxy() permet d'indiquer le passage par un relais

Sous-classes de URLConnection (2)

- **JarURLConnection** (classe abstraite)
  - Accès à une archive Java (Jar) locale ou distante
  - URL de la forme **jar:<url>!/<entry>** par exemple: jar:http://www.server.org/archive.jar!/dir/TheClass.class
  - Méthodes d'accès aux différentes parties et aux différents attributs des entrées de l'archive
  - Permet de récupérer des objets de la classe java.util.jar.JarEntry
  - Permet d'obtenir un flot de lecture sur l'entrée spécifiée par l'URL

Authentification

- Concerne les champs d'en-tête Authorization ou Proxy-Authorization lors de réponses
  - 401 Unauthorized ou 407 Proxy Authentication Required.
- Classe abstraite Authenticator
  - Gestionnaire d'authentification, installé par la méthode statique setDefault(Authenticator a)
  - Une fois installé, chaque requête reçevant une réponse 401 ou 407 fait appel, via l'objet URL, à la méthode statique requestPasswordAuthentication() qui appelle la méthode protégée getPasswordAuthentication() qui retourne un objet de classe PasswordAuthentication

Authentification (2)

- Classe PasswordAuthentication représente
  - Un identificateur (String)
  - Un mot de passe (char [])
- Par défaut getPasswordAuthentication() retourne null
  - Redéfinir cette méthode dans une sous classe concrète de Authenticator d'une méthode
  - Utiliser les méthodes getRequestingSite() (machine), getRequestingProtocol(), getRequestingScheme(), getRequestingPrompt() (realm du challenge), getRequestingPort() de la classe Authenticator
  - L'objet URL place ces valeurs dans les champs d'en-tête

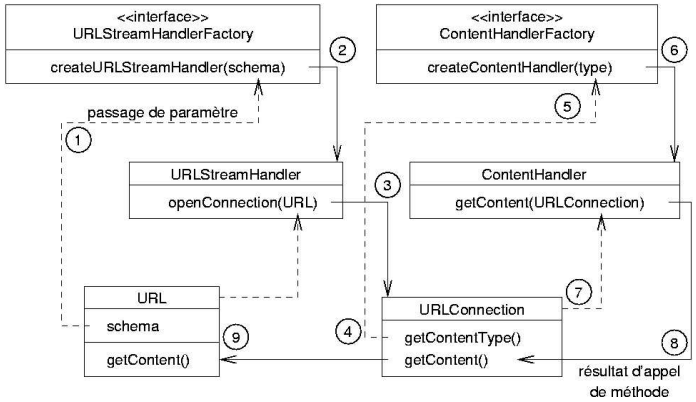
Gestionnaires

- Architecture ouverte autour des URL pour ajouter:
  - De nouveaux schémas d'URL
  - De nouveaux types de ressources reconnus
  - Ses propres implantations des gestionnaires existants
- Les classes d'objets concernés sont:
  - Gestionnaires de schémas (URLStreamHandler)
    - Analyse de l'URL et création du gestionnaire de connexion
  - Gestionnaires de connexion (URLConnection)
    - Gérer l'accès à la ressource et à ses attributs *via* un protocole
  - Gestionnaires de contenus (ContentHandler)
    - Transforme le flot sur la ressource en un objet Java

Gestionnaires par défaut

- Gestionnaire de schéma créé dynamiquement
  - à la première création d'URL ayant ce schéma <s>
  - sun.net.www.protocol.<s>.Handler
  - MalformedURLException si aucune classe ne correspond
- Gestionnaire de schéma détermine la classe des gestionnaires de connexion
  - openConnection() retourne un gestionnaire de connexion
- Gestionnaire de contenu créé par gestion de connexion
  - en fonction du type MIME <type>/<soustype> de la ressource
  - à sa première rencontre, lorsque getContent() est appelée
  - sun.net.www.content.<type>.<soustype>
  - UnknownServiceException si aucune classe trouvée de ce nom

Mécanisme gestionnaires



Mise en place de ses propres gestionnaires

- Pour utiliser son propore gestionnaire de schéma
  - Le passer en argument du constructeur de la classe URL
  - Sinon, installer une fabrique, implantant l'interface URLStreamHandlerFactory, par la méthde setURLStreamHandlerFactory() de la classe URL.
  - Elle retourne l'objet URLStreamHandler correspondant au protocole qu'elle sait traiter, ou sinon doit retourner null pour que le schéma par défaut s'applique
  - Sa méthode openConnection() retourne un gestionnaire de connexion spécifique, sous-classe de URLConnection
- Même principe existe pour les gestionnaires de contenu
  - URLConnection uc = ...;
    uc.setContentHandlerFactory(ContentHandlerFactory chf);

Applettes

- Délocaliser l'exécution de programmes sur le client
- Spécifié dans une page HTML (balises <applet>)
- C'est le navigateur du client qui exécute le bytecode
- Participe à l'*équilibrage de charge (load balancing)*
- Utilise les mécanismes de chargement de classe à distance
  - URLClassLoader
  - Permet de trouver et charger une classe parmi un ensemble d'URL de base, locaux ou distants

Chargement de classe

- Pour utiliser une classe (ou interface), la machine virtuelle doit avoir *chargé* son bytecode
- C'est un chargeur de classe qui s'en occupe
  - Classe abstraite java.lang.ClassLoader
- A toute classe A est associé un chargeur
  - Celui-ci est responsable du chargement des classes qui sont utilisées par A, et transitivement...
  - Possibilité de déléguer le chargement à un autre chargeur
  - Possibilité de demander si la classe n'a pas déjà été chargée

Chargement de classe (2)

- Pour la machine virtuelle, chaque classe chargée est identifiée par:
  - Son nom complet (avec package, ex: java.lang.String)
  - Son chargeur (celui qui a effectué le chargement du bytecode)
  - IMPORTANT: deux classes différentes de même nom complet peuvent cohabiter dans la machine virtuelle si elles n'ont pas été chargées par le même chargeur
- En plus, une politique de sécurité est associée à chacune de ces classes chargées
  - Idée: même degré de sécurité pour des classes provenant du même endroit

Chargeur de bootstrap

- Les objets chargeurs de classes sont des instances d'une sous-classe de ClassLoader
  - Le bytecode de cette sous-classe doit avoir été chargé...
  - Ce mécanisme récursif se termine par un chargeur de classe particulier dit ***bootstrap classloader***
- La méthode Class getClassLoader() appliquée sur un objet chargé par le bootstrap classloader retourne null
  - Sinon, elle retourne l'instance de ClassLoader qui a chargé la classe ayant créé cet objet

Trois groupes de classes

- Les classes de bootstrap
  - Nécessaires à l'environnement d'exécution standard
  - Distribuées par Sun dans la plate-forme standard et supportées dans le futur. Ex: les classes de **rt.jar**
  - Fichiers d'archives spécifiés par sun.boot.class.path
- Les classes de bibliothèque d'extensions standard
  - Fichiers d'archives spécifiés par java.ext.dirs
- Les classes définies par l'utilisateur
  - Par défaut, seul le répertoire courrant, mais on peut spécifier avec l'option -classpath ou variable d'environnement CLASSPATH (java.class.path)

La classe ClassLoader

- Par défaut, une classe est d'abord recherchée parmi les classes de bootstrap, puis parmi les classes d'extension et enfin seulement parmi les classes utilisateur
- Les chargeurs de classes sont organisés hiérarchiquement => notion de délégation
  - A chaque demande de chargement, un chargeur de classe délègue le chargement de la classe à son chargeur de classe parent, avant de tenter de le faire réellement lui-même en cas d'échec du chargeur parent

Les méthodes de ClassLoader

- Class loadClass(String name)
  - Demande au chargeur d'initier le chargement d'une classe (via des délégations éventuelles)
- Class findLoadedClass(String name)
  - Demande au chargeur s'il a déjà réellement chargé une classe
- Class findClass(String name)
  - Demande au chargeur de charger réellement une classe (sans aucune délégation à un autre chargeur)
- ClassLoader getParent()
- void resolveClass(Class c)
  - Prépare l'instance c pour son utilisation prochaine, fait des vérifications de bytecode, prépare et alloue des structures, réalise l'édition de liens

Schéma de délégation

- Le chargeur de classe, via sa méthode loadClass(), commence par rechercher, grâce à la méthode findLoadedClass(), s'il n'a pas déjà chargé la classe.
- S'il ne l'a pas déjà réellement chargée, il tente de déléguer le chargement au chargeur de classe parent grâce à getParent()
- Si le chargeur de classe parent n'a pas trouvé la classe, alors le chargeur courrant appelle la méthode findClass()
- En cas d'échec au final, une exception ClassNotFoundException est levée

Pseudo-code de chargement de classe

```
public Class loadClass(String name, boolean resolve)
  throws ClassNotFoundException {
  // Recherche de la classe parmi les classes déjà chargées
  Class c = findLoadedClass(name);
  if (c != null) return c;
  try {
    // Récupère le chargeur parent (référence supposée non nulle)
    ClassLoader parent = getParent();
    c = parent.loadClass(name); //Recherche avec le chargeur parent
    if (c != null) return c;
  } catch (ClassNotFoundException e) {} // Le parent n'a pas trouvé
    /* Si le chargeur parent n'a pas chargé la classe, c'est au
       chargeur courant de le faire. Si une RuntimeException est
       survenue pendant que le parent chargeait la classe, ou
       pendant le getParent(), ou bien encore pendant l'appel
       suivant, il faut la propager. */
  c = findClass(name);
  if (resolve) resolveClass(c);
  return c;
}
```


Modifier le comportement par défaut

- Pour modifier le comportement par défaut du chargement de classe
  - mais sans changer le schéma de délégation, il faut redéfinir dans son propre chargeur de classe la méthode findClass(), et ne pas modifier la méthode loadClass()
  - La méthode findClass() se charge de construire un objet de la classe Class grâce à la méthode Class defineClass(String name, byte[] b, int off, int len, ProtectionDomain domain) throws ClassFormatError
  - La méthode resolveClass() finira de préparer la classe chargée en vue de son utilisation

Création d'objet et exécution de code

- Une fois la classe obtenue et chargée par *le chargeur de classe* (celui qui a appelé la méthode defineClass())
  - On peut en créer des instances (newInstance(), par exemple)
  - Toutes les classes requises pour exécuter du code sur l'objet ainsi obtenu seront chargées par le même chargeur de classe
  - Il est ainsi possible que la même classe soit chargée par deux chargeurs différents: les deux classes chargées sont incompatibles

Sous-classes de ClassLoader

- SecureClassLoader (hérite de ClassLoader)
  - Utilise la politique de sécurité courante pour définir le domaine de sécurité en fonction d'un objet de la classe CodeSource (encapsule les notions d'URL de base et de certificats des signataires du code)
- URLClassLoader (hérite de SecureClassLoader)
  - Définit en plus une stratégie de recherche des classes parmi une liste d'URL correspondant
    - à des répertoires si les URL se terminent par des « / »
    - ou à des archives JAR dans tous les autres cas

URLClassLoader

- Constructeur le plus général
  - URLClassLoader(URL[] urls, ClassLoader parent, URLStreamHandlerFactory fabrique)
- En plus des méthodes loadClass(), findClass(), etc... plusieurs méthodes permettent de manipuler l'objet chargeur de classe relativement aux ressources qu'il peut désigner
  - URL findResource(String nom)
  - Enumeration findResources(String nom)
  - URL[] getURLs()

java.applet.Applet

- Recherche, chargement, exécution et contrôle de l'applette sont à la charge du navigateur
- Le programmeur dispose de quatre méthodes
  - **init()** appelée après le chargement et l'initialisation (conf)
  - **start()** appelée après **init()**, puis chaque fois que la fenêtre redevient visible
  - **stop()** appelée chaque fois que la fenêtre est cachée, pour suspendre l'animation par exemple
  - **destroy()** appelée pour terminer le cycle de vie de l'applette et libérer les ressources
  - Ces méthodes ne font rien par défaut, sauf isActive()

Graphique et navigateur

- Graphique
  - java.applet.Applet hérite de java.awt.Panel
  - Pour swing javax.swing.JApplet hérite de Applet
- Dans un document HTML,
  - entre les balises <applet> et </applet>, par exemple:
  - <applet code="NomDeClasse" width=150 height=80>
    <param name="NomParamètre" value="Valeur">
    </applet>
  - Éventuellement, spécifier grâce à codebase l'URL où trouver le bytecode si localisation différente
  - getCodeBase() *versus* getDocumentBase()

Souche et Contexte

- Une applette peut interagir avec le navigateur
  - setStub() est appelée par le navigateur à la création de l'applette, pour positionner une « souche » (AppletStub)
    Cette souche sert d'interface entre le navigateur et l'applette
    - isActive(), getAppletContext(), getCodeBase(), getDocumentBase(), getParameter(), etc...
  - getAppletContext() retourne un objet représentant le contexte d'exécution de l'applette (AppletContext).
    C'est le même objet contexte pour toutes les applettes d'un même document HTML
    - Possibilité de les récupérer via getApplets()

Droits des applettes

- Par défaut, les navigateurs interdisent
  - L'accès au système de fichier local (lecture et écriture)
  - Le lancement de processus au moyen de System.exec()
  - Le chargement de bibliothèque ou la définition de méthodes natives
  - L'accès ou la modification de propriétés systèmes donnant des informations sur l'utilisateur ou la machine locale (ex: user.name)
  - L'accès à un autre groupe de processus légers
  - L'installation de ClassLoader ou Factory
  - La connexion vers une autre adresse Internet que celle d'où provient l'applette
  - L'acceptation de nouvelles connexions
- Sinon, il faut signer l'applet et utiliser la politique de sécurité