- ACID : Atomicity, Consistency, Isolation, Durability // Atomicité, Cohérence, Isolation et Durabilité
- CRUD : Create, read, update and delete
- DAP : Domain Application Protocol

- SOLID
  Facilitent la conception générale de systèmes faciles à mainteniret à étendre (code fiable et robuste)
  Principes SOLID : identifiés par Robert Martin («Uncle Bob»)
  - S (Single responsability) : un objet fait une seule chose et est le seule à le faire
  - O (Open-Close) : classe ouverte aux extensions et fermées aux modifications
  - L (Liskov substitution) : classe fille doit pouvoir être utilisée à la place de sa mère (par une autre classe) en toute transparence
  - I (Interface Segregation) : plusieurs petites interfaces plutôt qu'une grosse
  - D (Dependency Inversion) : dépendre d'abstractions et pas d'implantations → essentiel pour permettre le test unitaire

  Robert Martin. « Design Principles and Design Patterns ». objectmentor.com

- "Java8+"
  - 5 : types génériques, metadata / annotations ; autoboxing entre types primitifs ; énumérations, varargs (Object... / tableaux) ; "foreach"
  - 6 : correctifs...
  - 7 : notation binaire, formatage numérique, switch string, inférence types génériques à la création, multicatch exceptions, java.nio (Files, Path...), autoboxing des primitifs...
  - 8 : lambda, stream, interfaces fonctionnelles
  - 9 : jigsaw (modularisation) ; kulla (JShell) ; JSON natif et http2
  - 10 : inférence types "var", Graal / JIT plus rapide
  - 11 : améliorations lambda, surpression CORBA

- Open Web Application Security Project (OWASP) : sécurité des applications Web

- Lambda
  Programmation Fonctionnelle, se passer (au moins partiellement) de l'aspect "machine états" de la POO
  Syntaxe : *(parametres) -> Corps de la fonction*
  cas particuliers de implicite (types inférables), pas inner class => pas nouveau scope ni référence mutable...
  crée méthode static dans classe où défini
  raccourci pour implémentation interface fonctionnelle (cas génériques déjà créés : java.util.function)
  Supplier<T> // Consumer<T> // Function<T, R> // Predicate<T>
  Raccourcis syntaxiques, 4 cas possibles :
  méthodes statiques
  constructeur
  méthode d'instance
  méthode objet arbitraire (type particulier)
  `qualificateur::methode (type ou variable :: methode)`
- Optional => PAS EN PARAMÈTRE ! *ajouté dans JavaDoc entre Java8 et Java9*
- Stream (i.e. Pipeline / Workflow de traitement de données)
  Déclaratif et non purement procédural (idée du "Builder de Pipeline")
  Opérations intermédiaires / Opérations terminales ; Opérations <stateful / stateless> (mémorisation valeurs ou pas)
  Opérations intermédiaires : transformations (en sortie), collection d'origine ne doit pas être modifiée
  limit / skip / distinct
  peek / filter / sort
  map / flatMap (transformation)
  Opérations terminales : lance traitement et ne permet plus réutilisation du Stream
  *Match (noneMatch, anyMatch, allMatch)
  find* (findAny, findFirst)
  min, max, count
  reduce / forEach
  collect / toArray
  Spliterator : *utile pour paralléliser (plusieurs itérateurs de traitement)*
  Collectors
  supplier // accumulator // combiner // finisher

MAVEN

— Structuration des répertoires

```
src
    main
        java
        resources
    test
        java
        resources
target
```

— pom.xml : là où est décrit le projet maven
— <build> : décrit les fonctions relatives à la construction d'1 projet (classpath, version de java...)
— <profiles> : profils de construction, pour personnaliser la totalité du pom selon des identifiants de profil
— <dependencies> : liste les dépendances selon 1 format « qualifié »
  — groupId : nom d'un groupe (en général : préfixe du package principal)
  — artifactId : nom de la dépendance (en général : nom du projet)
  — version : version du projet–type : format (jar, pom...)
  — scope (périmètre/portée de la dépendance) :
    - à la compilation (compile), uniquement à l'exécution (runtime),
    - pendant la compilation et l'exécution des tests (test),

— Phases du cycle de vie du projet / Utilitaire en ligne de commande, à lancer depuis l'emplacement du fichier pom.xml :
— mvn [goal]avec [goal] = (invoque les plugins nécessaire pour faire le travail)

generate-sources génère le code source supplémentaire nécessité par l'appli (accompli par les plugins appropriés)
clean supprimer les éléments précédemment construits
compile compile le code source du projet
test-compile compile les tests unitaires du projet
test lancer/exécuter les tests unitaires (avec Junit)
package packager le code compilé (un WAR, un JAR)
install installer le JAR sur le dépôt local
deploy déployer l'application sur le serveur cible (pour être partagé avec d'autres développeurs

REST (Representational state transfer)

— Orienté RESSOURCES ; différent de SOAP (services)
— Contraintes architecturales
  — Client–serveur
  — Sans état
  — Avec mise en cache
  — En couches
  — Avec code à la demande (facultative)
  — Interface uniforme
— Following four HTTP methods are commonly used in REST based architecture.

GET – Provides a read only access to a resource.
POST – Used to create a new resource.
PUT – Used to update a existing resource or create a new resource.
DELETE – Used to remove a resource.

équivalents SQL : SELECT / INSERT / UPDATE / DELETE

GET C'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il doit être possible de répéter la requête sans effet.
HEAD Cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.
POST Cette méthode est utilisée pour transmettre des données en vue d'un traitement à une ressource (le plus souvent depuis un formulaire HTML). L'URI fourni est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes. À cause de la mauvaise implémentation des méthodes HTTP (pour Ajax) par certains navigateurs (et la norme HTML qui ne supporte que les méthodes GET et POST pour les formulaires), cette méthode est souvent utilisée en remplacement de la requête PUT, qui devrait être utilisée pour la mise à jour de ressources.
OPTIONS Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.
CONNECT Cette méthode permet d'utiliser un proxy comme un tunnel de communication.
TRACE Cette méthode demande au serveur de retourner ce qu'il a reçu, dans le but de tester et effectuer un diagnostic sur la connexion.
PUT Cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.
PATCH Cette méthode permet, contrairement à PUT, de faire une modification partielle d'une ressource.
DELETE Cette méthode permet de supprimer une ressource du serveur.

Memento MAVEN

- **POM** : Project Object Model (descripteur de projet maven)
- **Artefact** : fichier résultat du build d'un projet
- **Groupe** : Espace de nom permettant le regroupement d'artefacts
- **Version** : identification d'une version d'un artefact
- **Repository** : référentiel (dépôt) d'artefacts
- **Plugin** : composant enfichable dans le processus de build
- **Archétype** : modèle (prototype) de projet
- **Goal** : objectif du build, réalisé par un plugin
- **Phases : étapes clés de la construction du projet**
  1. validate : validation du projet, voir s'il est correct
  2. compile : compilation du projet
  3. test : compiler les tests unitaires et les passer
  4. package : mettre en forme le livrable. Souvent : faire un jar/un war
  5. integration-test : Tests qui doivent passer sur un environnement
  6. verify : Vérifier la qualité du livrable suivant une série de critères
  7. install : Déposer le package livrable sur le Repository Local
  8. deploy : Publier le package final sur un environnement partagé
- hors cycle de vie : clean, site, assembly :assembly, site-deploy...
- **Convention plutôt que configuration**
  - /src : les sources du projet
  - /src/main : code source et fichiers source principaux
  - /src/main/java : code source
  - /src/main/resources : fichiers de ressources (images, annexes...)
  - /src/main/webapp : webapp du projet
  - /src/test : fichiers de test
  - /src/test/java : code source de test
  - /src/test/resources : fichiers de ressources de test
  - /src/site : informations sur le projet et/ou les rapports générés suite aux traitements effectués
  - /target : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests

- **pom.xml** `http://maven.apache.org/maven-model/maven.html`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example.package</groupId>
    <artifactId>ApplicationName</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>

    <name>Project Name</name>
    <url>http://www.example.org/</url>

    <dependencies>
        <!-- ... -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- ... -->
    </dependencies>

    <properties>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.compiler.source>1.8</maven.compiler.source>
        <build.name>BuildName</build.name>
        <!-- ... -->
    </properties>

    <build>
        <finalName>${build.name}</finalName>
        <plugins>
            <!-- ... -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>
            <!-- ... -->
        </plugins>
    </build>
</project>
```

Memento REST

- Services Web
  - Protocoles standardisés / ouverts (en général HTTP)
  - Deux familles : SOAP (orienté services) et REST (orienté ressources)
  - REpresentational State Transfer, chapitre 5 thèse de Roy Fielding (2000), systèmes distribués, `http://opikanoba.org/tr/fielding/rest`
  - Architecture, ensemble de contraintes...
  - Client / serveur ; Stateless ; interface uniforme (identification, représentation, auto-descriptif, HATEOAS / Hypermedia) ; couches ; code à la demande (optionnel)
- Client / serveur : sépare IHM et stockage (évolution séparée, meilleure portabilité et possibilités montée en charge...)
- Stateless : requête auto-portante ; pas de contexte sur le serveur (côté client) ; améliore visibilité
- Données réponses marquées pour la mise en cache
- Interfaces uniformes (standardisé) : cf. SQL / SGBDR (schema.table.colonne) : SELECT, INSERT, UPDATE et DELETE
- Contraintes d'interface : identification uniforme (URI : Uniform Resource Identifier)...
- Représentations (seule la représentation de l'état transite)
- Hypermédia comme moteur de l'état de l'application (HATEOAS : Hypertext As The Engine Of Application State) : facilité, guidage utilisateur, agent utilisateur stocke l'information, pas de contraintes pré-définies...
- ...
- HTTP / URI = Uniform Resource Identifier (RFC 3986) `https://tools.ietf.org/html/rfc3986`
  - URL : Uniform Resource Locator (identifie une ressource réseau)
  - URN : Uniform Resource Name (identifie un nom de ressource)
  - Schema + Chemin d'accès
- Requête : Structure textuelle normalisée
  - Ligne de requête : <méthode> <URI> <version-HTTP> CRLF
  - Lignes d'en-tête : *(<champs> : <valeur> CRLF)
  - Ligne vide : CRLF
  - Corps (optionnel) : [*OCTET]
- Réponse : Structure assez similaire à la requête
  - Ligne de statut : <version-HTTP> <statut> <message> CRLF
  - Lignes d'en-tête : *(<champs> : <valeur> CRLF)
  - Ligne vide : CRLF
  - Corps (optionnel) : [*OCTET]
- De nombreux champs d'en-tête sont prédéfinis
  - General header : les champs communs (Cache-Control, Date, ...)
  - Request header : les champs spécifiques de requête (Host, Accept, ...)
  - Response header : les champs spécifiques de réponse (Age, ETag, ...)
  - Entity header : les champs spécifiques des entités (Content-Encoding, ...)
- Spécifications : http ://www.w3.org/Protocols/
  - HTTP 1.1 : https ://tools.ietf.org/html/rfc2616
  - HTTP 1.1bis : reformulation dans plusieurs documents (RFC7230, . . . , RFC7237)
  - HTTP 2 : RFC7540 publiée en mai 2015
- 8 méthodes de base
  - GET : obtenir une représentation d'une ressource
  - HEAD : demander des informations sur une ressource
  - POST : transmettre des données à une ressource pour traitement
  - PUT : remplacer ou créer une ressource (dont on connaît l'URI)
  - DELETE : supprimer une ressource
  - CONNECT : pour utiliser un Proxy comme un tunnel de communication
  - OPTIONS : demander les options de communications (ressource ou serveur)
  - TRACE : demander au serveur de retourner ce qu'il a reçu (echo pour diagnostic)
- Codes de réponse HTTP :

  1xx – Information messages d'information (non utilisé)
  2xx – Succès la requête s'est déroulée correctement
  3xx – Redirection indique une redirection
  4xx – Erreur client à l'origine du client
  5xx – Erreur serveur à l'origine du serveur

Table des matières

...
some data...