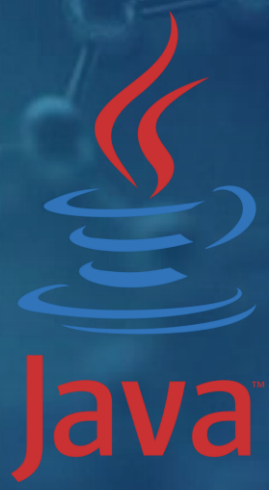
Java Inside TP1

ZEYNALI KYAN – E4 INFO ADD

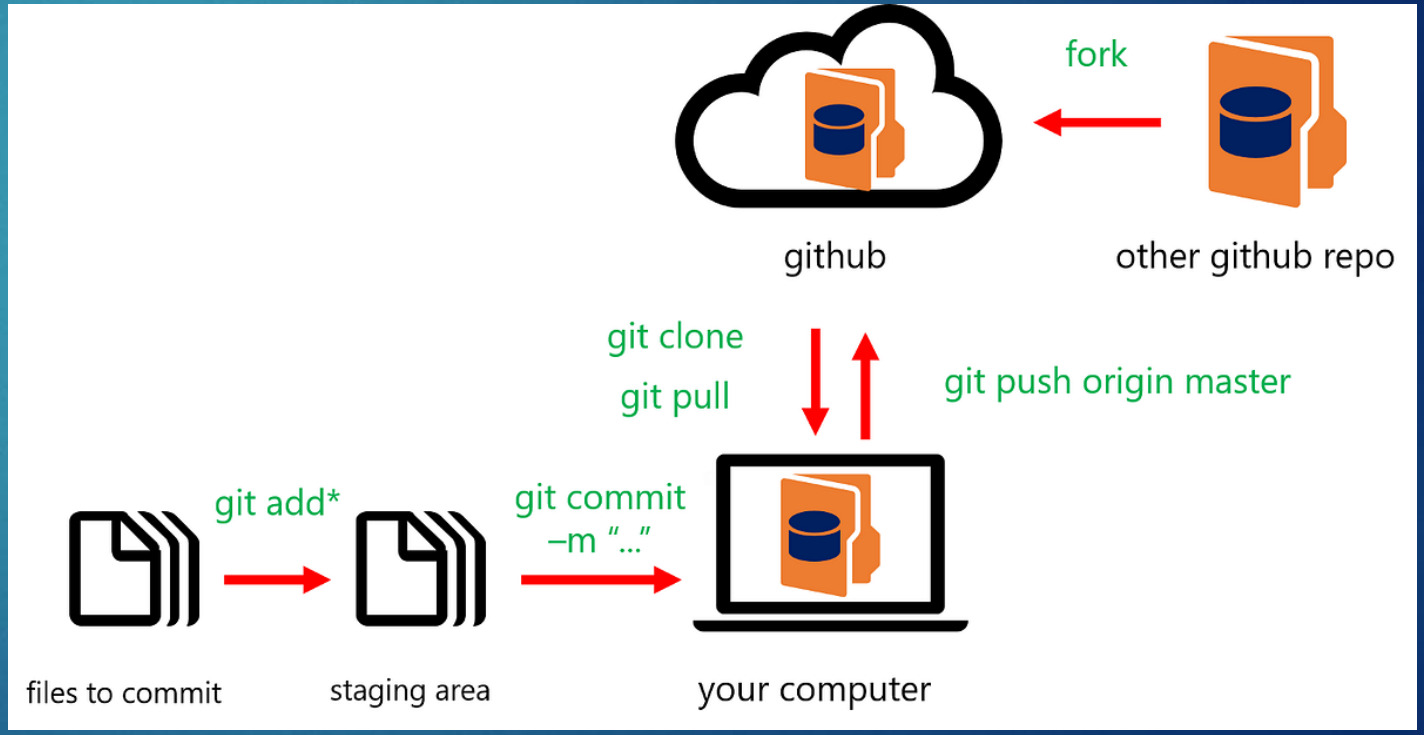
10 SEPTEMBRE 2024



Git et GitHub

GitHub

- Gestion de versions
- Collaboration
- Réversibilité et Distribution



Nouveautés des switch sur les types

```
public String toJSON(Object o) {
  return switch (o) {
    case null -> "null";
    case Boolean b -> "" + b;
    case Integer i -> "" + i;
    case Long l -> "" + l;
    case Float f -> "" + f;
    case Double d -> "" + d;
    case String s -> '"' + s + '"';
    default -> throw new IllegalArgumentException();
  };
}
```

- Case sur la valeur null
- Raccourci pour print des valeurs
- Variables locales anonymes avec Under scores _ (java 22)
- Default uniquement si on match un Object, la classe mère

Java beans et la classe BeanInfo

```java
// Java bean for Person
public class Person {
    // Private variables
    private String firstName;
    private String lastName;

    // Getters and setters for variables
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName (String firstName) {
        this.firstName=firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName (String lastName) {
        this.lastName=lastName;
    }
}
```

ATTENTION : La classe en elle-même (.class) est incluse dans les propriétés d'un BeanInfo

- Utile pour la sérialisation pour permettre à un objet Java d'être sauvegardé et restauré
- Interface fournissant des informations utiles sur les JavaBeans via la réflexion
- Méthodes principales de BeanInfo :
  - getPropertyDescriptors()
  - getMethodDescriptors()
  - getEventSetDescriptors()
  - getBeanDescriptor()
- Méthode getPropertyDescriptors() très couteuse en mémoire

Avant-goût des classes internes

```
public final class JSONWriter {

  private record BiduleGenerator(String prefix, Method getter)
      implements Generator {
    @Override
    public String generate(JSONWriter writer, Object bean) {
      return prefix + writer.toJSON(Utils.invokeMethod(bean, getter));
    }
  }

  @FunctionalInterface
  private interface Generator {
    String generate(JSONWriter writer, Object bean);
  }

}
```

S'applique à tous les types de classe :
Classes, Records, Interfaces, Classes Abstraites, Enums, …

- **Classe interne classique** : Classe définie à l'intérieur d'une autre et qui a accès aux membres de la classe englobante
- **Classe interne statique** : Classe ne nécessitant pas une instance de la classe englobante pour être instanciée
- **Classes locales** : Classes définies à l'intérieur d'un bloc de code (i.e. Méthodes)
- **Classes anonymes** : Classes sans nom, souvent utilisées pour implémenter des interfaces ou des classes abstraites.

Lambdas et classes anonymes

```
private static final ClassValue<List<Generator>> BEAN_INFO_CLASS_VALUE = new ClassValue<>() {
  @Override
  protected List<Generator> computeValue(Class<?> type) {
    return Arrays.stream(Utils.beanInfo(type).getPropertyDescriptors())
      .filter(property -> !"class".equals(property.getName()))
      .<Generator>map(property -> {
        var getter = property.getReadMethod();
        var annotation = getter.getAnnotation(JSONProperty.class);
        var name = annotation == null ? property.getName() : annotation.value();
        var prefix = '"' + name + "\": ";
        return (writer, bean) -> prefix + writer.toJSON(Utils.invokeMethod(bean, getter));
        // return new BiduleGenerator(prefix, getter);
      })
      .toList();
  }
};
```

Avantages des lambdas par rapport aux classes anonymes :

- Lisibilité du code (moins verbeux)
- Syntaxe simple similaire aux langages fonctionnels
- Récupération des variables du bloc de code englobant
- Prétraitement simplifié et gain de performances sur les opérations redondantes

Annotations personnalisées

```java
@Retention(RUNTIME)
@Target({METHOD, RECORD_COMPONENT})
public @interface JSONProperty {
  String value();
}
```

- Métadonnées fournissant des informations supplémentaires dans la Javadoc
- Dans le JDK de base n'a aucun effet sur l'exécution
- Utilisées par les frameworks pour générer du code (ex : Spring)
- Nouveau mot-clé de fichier java : @interface => Attention aux faux-ami
- Annotations communes : @Override , @Deprecated , @FunctionnalInterface