```
public String getFullText() {
    return fullText;
}

@Lob
public byte[] getFullCode() {
    return fullCode;
}
```

Si le type de la propriété implémente `java.io.Serializable` et n'est pas un type de base, et si la propriété n'est pas annotée avec `@Lob`, alors le type Hibernate `serializable` est utilisé.

2.2.2.2. Déclarer des attributs de colonne

La(les) colonne(s) utilisée(s) pour mapper une propriété peuvent être définies en utilisant l'annotation `@Column`. Utilisez-la pour surcharger les valeurs par défaut (voir la spécification EJB3 pour plus d'informations sur les valeurs par défaut). Vous pouvez utilisez cette annotation au niveau de la propriété pour celles qui sont :

- pas du tout annotées
- annotées avec `@Basic`
- annotées avec `@Version`
- annotées avec `@Lob`
- annotées avec `@Temporal`
- annotées avec `@org.hibernate.annotations.CollectionOfElements` (pour Hibernate uniquement)

```
@Entity
public class Flight implements Serializable {
...
@Column(updatable = false, name = "flight_name", nullable = false, length=50)
public String getName() { ... }
```

La propriété `name` est mappée sur la colonne `flight_name`, laquelle ne peut pas avoir de valeur nulle, a une longueur de 50 et ne peut pas être mise à jour (rendant la propriété immuable).

Cette annotation peut être appliquée aux propriétés habituelles ainsi qu'aux propriétés `@Id` ou `@Version`.

```
@Column(
    name="columnName";                                  (1)
    boolean unique() default false;                     (2)
    boolean nullable() default true;                    (3)
    boolean insertable() default true;                  (4)
    boolean updatable() default true;                   (5)
    String columnDefinition() default "";               (6)
    String table() default "";                          (7)
    int length() default 255;                           (8)
    int precision() default 0; // decimal precision     (9)
    int scale() default 0; // decimal scale
```

(1) `name` (optionnel) : le nom de la colonne (par défaut le nom de la propriété)
(2) `unique` (optionnel) : indique si la colonne fait partie d'une contrainte d'unicité ou non (par défaut false)
(3) `nullable` (optionnel) : indique si la colonne peut avoir une valeur nulle (par défaut false).

(4) `insertable` (optionnel) : indique si la colonne fera partie de la commande insert (par défaut true)
(5) `updatable` (optionnel) : indique si la colonne fera partie de la commande update (par défaut true)
(6) `columnDefinition` (optionnel) : surcharge le fragment DDL sql pour cette colonne en particulier (non portable)
(7) `table` (optionnel) : définit la table cible (par défaut la table principale)
(8) `length` (optionnel) : longueur de la colonne (par défaut 255)
(8) `precision` (optionnel) : précision décimale de la colonne (par défaut 0)
(10) `scale` (optionnel) : échelle décimale de la colonne si nécessaire (par défaut 0)

2.2.2.3. Objets embarqués (alias composants)

Il est possible de déclarer un composant embarqué à l'intérieur d'une entité et même de surcharger le mapping de ses colonnes. Les classes de composant doivent être annotées au niveau de la classe avec l'annotation `@Embeddable`. Il est possible de surcharger le mapping de colonne d'un objet embarqué pour une entité particulière en utilisant les annotations `@Embedded` et `@AttributeOverride` sur la propriété associée :

```
@Entity
public class Person implements Serializable {

    // Composant persistant utilisant les valeurs par défaut
    Address homeAddress;

    @Embedded
    @AttributeOverrides( {
            @AttributeOverride(name="iso2", column = @Column(name="bornIso2") ),
            @AttributeOverride(name="name", column = @Column(name="bornCountryName") )
    } )
    Country bornIn;
    ...
}
```

```
@Embeddable
public class Address implements Serializable {
    String city;
    Country nationality; // par de surcharge ici
}
```

```
@Embeddable
public class Country implements Serializable {
    private String iso2;
    @Column(name="countryName") private String name;

    public String getIso2() { return iso2; }
    public void setIso2(String iso2) { this.iso2 = iso2; }


    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    ...
}
```

Un objet embarquable hérite du type d'accès de son entité d'appartenance (notez que vous pouvez surcharger cela en utilisant les annotations spécifiques à Hibernate `@AccessType`, voir Extensions d'Hibernate Annotation).

L'entity bean `Person` a deux propriétés composant, `homeAddress` et `bornIn`. La propriété `homeAddress` n'a pas été annotée, mais Hibernate devinera que c'est un composant persistant en cherchant l'annotation `@Embeddable`