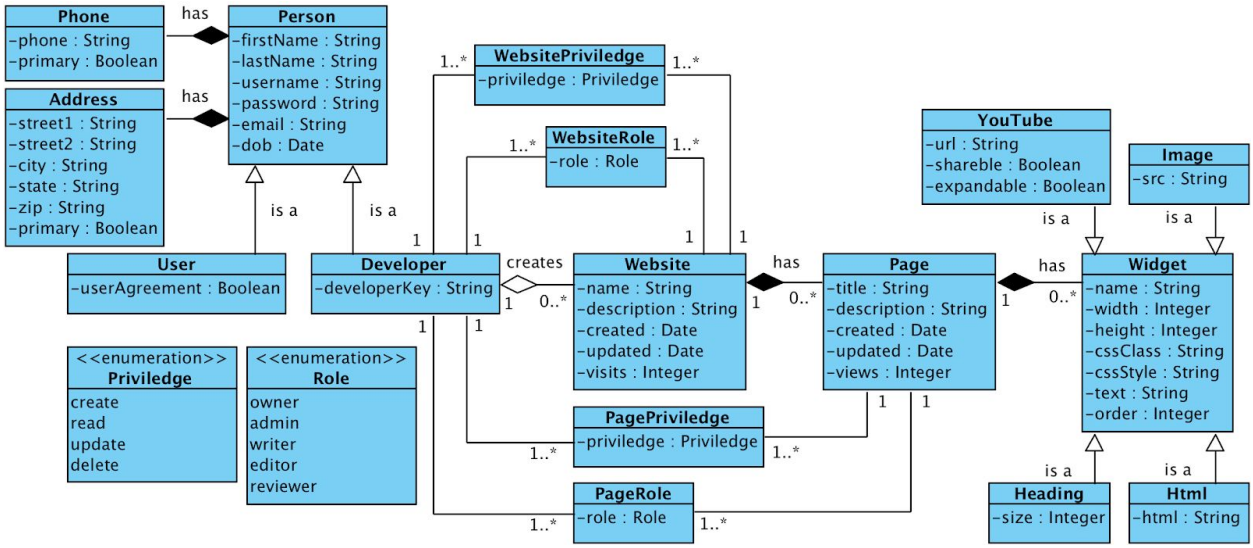
JDBC Assignment

CS5200 - Introduction to Relational Databases

Consider the UML class diagram shown below. Create the corresponding Java Data Model that implements the equivalent relational model, fulfills the use cases, implements the relations, and enforces the constraints. Implement Data Access Objects (DAOs) that encapsulate access to the database applying best practices discussed in class. A link to this assignment can be found here.

UML Class Diagram

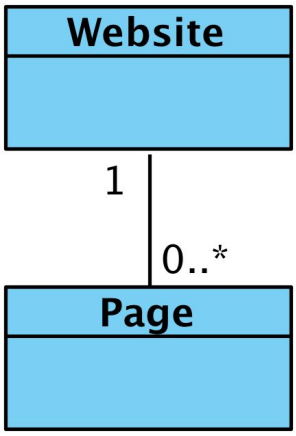
Consider the class diagrams below



The UML diagram defines several associations between various classes using lines and symbols. These are implemented in SQL using primary and foreign key fields. In Java you'll also capture the primary key fields id, but instead of the foreign key fields, you'll capture some collection of instances of the child class. As an example, consider the relation between entities Website and Page illustrated below. The UML class diagram captures the association using a line and the cardinality of how many of each participate in relation. The diagram says that each Website instance has zero or more Page instance.

SQL implements the relation using primary and foreign keys. In the example, the **Page** table declares foreign key **websiteId** which references to the **id** field in the **Website** table. If we consider the relation between websites and pages as a whole/parts relation, where pages are the various parts that make a whole Website, then the relation implementation states that each part (page) know who the whole is (website). The Java implementation captures the relation between websites and pages by declaring a collection of pages in the **Website** class. That is, each **Website** instance has a complete list of references to its **Page** instance. Notice the difference in the direction of the relation between the SQL and the Java relation implementation. Whereas in the SQL implementation the relation is from the part to the whole, e.g., page records have foreign key references to websites, in the Java implementation, the relation goes from the whole to the parts, e.g., the websites have a collection of references to page instances. Additionally, the Java implementation also can declare a reference from the page to the website which matches the SQL implementation, but this is less common in object oriented solutions.

UML	SQL	Java

	`CREATE TABLE Website (`
	`  id int,`
	`  PRIMARY KEY (id)`
	`);`
	`CREATE TABLE Page (`
	`  id int,`
	`  websiteId int,`
	`  FOREIGN KEY (websiteId)`
	`  REFERENCES Website(id)`
	`);`
		`class Website {`
		`  int id;`
		`  Collection<Page> pages;`
		`  addPage(Page) { }`
		`  removePage(Page) { }`
		`}`
		`class Page {`
		`  Website website;`
		`}`

A good naming convention for primary and foreign keys is to name all foreign keys "**`id`**" (or "**`ID`**") and name the foreign keys that refer to them as the table name followed by "**`Id`**". For instance, from the example above, the foreign key from table **`Page`** to table **`Website`** should be named **`websiteId`** and should refer to the primary key field **`id`** in table **`Website`**.

Create Java Data Model (5pts.)

Use Java to implement the data model. Make use of best practices discussed in class. Each instance variables must be declared as private and have public setters and getters.

1. **`Person.java`**
2. **`User.java`**
3. **`Developer.java`**
4. **`Website.java`**
5. **`Page.java`**
6. **`Widget.java`**
7. **`HeadingWidget.java`**
8. **`HtmlWidget.java`**
9. **`ImageWidget.java`**
10. **`YouTubeWidget.java`**
11. **`Role.java`**
12. **`Priviledge.java`**

Constructor Requirements: (5Pts)

Follow the constructor requirements specified below. There are no constructor requirements for classes not specified below.

1. **Developers.java**
    a. Should create an instance of developer when developer Key, Developer ID, Developer First Name, Developer Last Name, with rest of the fields with default values specified by you.
    b. Should also create an instance of developer when developer key, developer ID, developer first name, developer last name, developer user-name, developer password, email and dob are passed as parameters.
    c. Should also create an instance of developer when developer key, developer ID, developer first name, developer last name, developer user-name, developer password, email, dob, addresses, and phone numbers are passed as parameters.
2. **User.java**

    a. Should create an instance of user when user ID, First Name, Last Name are passed as parameters.
3. **Websites.java**
    a. Should create an instance of website when id, website name, description, created, updated, and visits are passed as parameters.
4. **Page.java**
    a. Should create an instance of Page when id, title, description, created, updated and views are passed as parameters.
5. **Widget.java**
    a. Should create an instance of Widget when ID, name, width, height, cssStyle, cssCLass, text, order are passed as parameters.

Naming Conventions (5pts)

These are the naming conventions for the project and they must be followed. Else your JUnit grader will fail and would not grade your assignments.

1. Your Projects must be named cs5200_fall2019_lastname_firstname_jdbc
2. All Packages must be created in src->main->java->edu->northeastern->cs5200
3. Inside your edu.northeastern.cs5200 package,
    a. All DAOs must be in package called *daos.*
    b. All Impl classes must be in the daos package.
    c. All models must be in package called *models*.
    d. If other packages are created, please specify their names in README.txt.

Implementation(5Pts)

*****All applications should be deployed in AWS, unless otherwise excused.*****

In src->main->java>edu>northeastern>cs5200 package, create a public Singleton class called Connection.java.

```
public class Connection {

    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://<AWS ENDPOINT>/<SCHEMA NAME>";
    private static final String USER = <AWS RDS USERNAME>;
    private static final String PASSWORD = <AWS RDS PASSWORD>;
    private static   java.sql.Connection dbConnection = null;

    public static Connection getConnection() throws ClassNotFoundException, SQLException {
    Class.forName(DRIVER);

    If (dbConnection == null) {
            dbConnection = DriverManager.getConnection(URL, USER, PASSWORD);
            return dbConnection;
    } else { return dbConnection; } }
```

```
        public static void closeConnection() {
         try {
                  If (dbConnection != null) {
                 dbConnection.close();
                 dbConnection = null; }
         } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
         }
        }
}
```

So, everytime the test method calls the static method called getInstance() of the Connection.java class, it should return the same instance of the connection rather t han creating a new instance. The class should have method called closeConnection(). It must be a static method and should not return anything. It closes the above said connection.

Create Data Access Objects (40pts.)

A good practice is to encapsulate all data access in a set of classes whose only responsibility is to provide an API to the database. These types of classes are commonly referred to as Data Access Objects, or DAOs. Implement the following DAOs that encapsulate the access to the database for each of the entities and relations in the UML class diagram.

1. **DeveloperDao.java**
2. **WebsiteDao.java**
3. **PageDao.java**
4. **WidgetDao.java**
5. **RoleDao.java**
6. **PriviledgeDao.java**

Implement the Developer Data Access Object

In a file called **DeveloperImpl.java**, create an interface shown below. Please do not make any modification to the interface.

1. **`void createDeveloper(Developer developer)`**
   inserts properties in **`developer`** instance parameter in tables **`Developer`** and **`Person`**
2. **`Collection<Developer> findAllDevelopers()`**
   returns all joined records from **`Developer`** and **`Person`** tables as a **`Collection`** of **`Developer`** instances.
3. **`Developer findDeveloperById(int developerId)`**
   returns a joined record from **`Developer`** and **`Person`** tables whose **`id`** field is equal to the **`developerId`** parameter
4. **`Developer findDeveloperByUsername(String username)`**
   returns a joined record from **`Developer`** and **`Person`** tables whose **`username`** field matches the parameter.
5. **`Developer findDeveloperByCredentials(String username, String password)`**
   returns a joined record from **`Developer`** and **`Person`** tables whose **`username`** and **`password`** fields match the parameters

6. **`int updateDeveloper(int developerId, Developer developer)`**
   updates records in **`Developer`** and **`Person`** tables whose **`id`** field is equal to **`developerId`** parameter. New record field values are set to the values in the **`developer`** instance parameter.
7. **`int deleteDeveloper(int developerId)`**
   deletes records from **`Developer`** and **`Person`** tables whose **`id`** field is equal to **`developerId`** parameter. *Do not make any modifications to the instance provided.*

In a file called **`DeveloperDao.java`**, develop a class which implements the Developer interface and encapsulates all database interaction between the **`Developer`** and **`Person`** entities and relations.

When returning the developer instance, create a new developer instance and do not modify the existing developer instance provided in the parameters.

Implement Website Data Access Object

In a file called **WebsiteImpl.java**, create an interface as shown below. Do not make any modifications to the interface.

1. **`void createWebsiteForDeveloper(int developerId, Website website)`**
   inserts properties in **`website`** instance parameter into the **`Website`** table. The website's **developerId** foreign key refer to **Developer** table primary key **id** whose value is equal to the **developerId** parameter. You can use the owner's user id as the foreign key
2. **`Collection<Website> findAllWebsites()`**
   returns all records from **`Website`** table as a **`Collection`** of **`Website`** instances
3. **`Collection<Website> findWebsitesForDeveloper(int developerId)`**
   returns all records from **`Website`** table as a **`Collection`** of **`Website`** instances whose **developerId** is equal to the **developerId** parameter
4. **`Website findWebsiteById(int websiteId)`**
   returns a record from **`Website`** table whose **`id`** field is equal to the **`websiteId`** parameter
5. **`int updateWebsite(int websiteId, Website website)`**
   updates record in **`Website`** table whose **`id`** field is equal to **`websiteId`** parameter. New record field values are set to the values in the **`website`** instance parameter
6. **`int deleteWebsite(int websiteId)`**
   deletes record from **`Website`** table whose **`id`** field is equal to **`websiteId`** parameter

In a file called **`WebsiteDao.java`**, implement a class that encapsulates all database interaction between the **`Website`** and **`Developer`** entities and relations.

When returning the website instance, create a new website instance and do not modify the existing developer instance provided in the parameters.

Implement Page Data Access Object

In a file called **PageImpl.java,** create an interface with the methods given below. Do not make any modifications.

1. **`void createPageForWebsite(int websiteId, Page page)`**
   inserts properties in **`page`** instance parameter into the **`Page`** table. The page's **`websiteId`** foreign key refer to **Website** table primary key **id** whose value is equal to the **`websiteId`** parameter
2. **`Collection<Page> findAllPages()`**
   returns all records from **`Page`** table as a **`Collection`** of **`Page`** instances
3. **`Page findPageById(int pageId)`**
   returns a record from **`Page`** table whose **`id`** field is equal to the **`pageId`** parameter
4. **`Collection<Page> findPagesForWebsite(int websiteId)`**
   returns all records from **`Page`** table as a **`Collection`** of **`Page`** instances whose **websiteId** is equal to the **websiteId** parameter
5. **`int updatePage(int pageId, Page page)`**
   updates record in **`Page`** table whose **`id`** field is equal to **`pageId`** parameter. New record field values are set to the values in the **`page`** instance parameter
6. **`int deletePage(int pageId)`**
   deletes record from **`Page`** table whose **`id`** field is equal to **`pageId`** parameter

In a file called **`PageDao.java`**, implement a class that encapsulates all database interaction between the **`Website`** and **`Page`** entities and relations. Use the interface shown below.

When returning the page instance, create a new page instance and do not modify the existing developer instance provided in the parameters.

Implement Widget Data Access Object

In a file called **WidgetImpl.java,** create an interface with the methods given below. Do not make any modifications.

1. **`void createWidgetForPage(int pageId, Widget widget)`**
   inserts properties in **`widget`** instance parameter into the **`Widget`** table. The widget's **`pageId`** foreign key refer to **`Page`** table primary key **`id`** whose value is equal to the **`pageId`** parameter
2. **`Collection<Widget> findAllWidgets()`**
   returns all records from **`Widget`** table as a **`Collection`** of **`Widget`** instances
3. **`Widget findWidgetById(int widgetId)`**
   returns a record from **`Widget`** table whose **`id`** field is equal to the **`widgetId`** parameter
4. **`Collection<Widget> findWidgetsForPage(int pageId)`**
   returns all records from **`Widget`** table as a **`Collection`** of **`Widget`** instances whose **pageId** is equal to the **pageId** parameter
5. **`int updateWidget(int widgetId, Widget widget)`**
   updates record in **`Widget`** table whose **`id`** field is equal to **`widgetId`** parameter. New record field values are set to the values in the **`widget`** instance parameter
6. **`int deleteWidget(int widgetId)`**
   deletes record from **`Widget`** table whose **`id`** field is equal to **`widgetId`** parameter

In a file called **`WidgetDao.java`**, implement a class that encapsulates all database interaction between the **`Widget`** and **`Page`** entities and relations.

When returning the widget instance, create a new widget instance and do not modify the existing developer instance provided in the parameters.

Implement Role Data Access Object

In an interface called RoleImpl.java, create an interface with methods given below. In a file called **`RoleDao.java`**, implement a class that encapsulates all database access that manage website and page roles assigned to **`Developer`**.

1. `assignWebsiteRole(int developerId, int websiteId, int roleId)`
   inserts into table Role a record that assigns a developer whose id is developerId, the role with roleId, to the website with websiteId
2. `assignPageRole(int developerId, int pageId, int roleId)`
   inserts into table Role a record that assigns a developer whose id is developerId, the role with roleId, to the page with pageId
3. `deleteWebsiteRole(int developerId, int websiteId, int roleId)`
   deletes from table Role a record that removes roleId from developerId, on websiteId
4. `deletePageRole(int developerId, int pageId, int roleId)`
   deletes from table Role a record that removes roleId from developerId, on pageId

When returning the role instance, create a new role instance and do not modify the existing role instance provided in the parameters.

Implement Priviledge Data Access Object

In a file called Privilege Impl.java, create an interface which defines the below methods. In a file called **`PriviledgeDao.java`**, implement a class that encapsulates all database access that manage website and page priviledges assigned to **`Developer`**.

1. **`assignWebsitePriviledge(int developerId, int websiteId, String priviledge)`**
   inserts into table Priviledge a record that assigns a developer whose id is developerId, the priviledge with priviledge name, to the website with websiteId
2. **`assignPagePriviledge(int developerId, int pageId, String priviledge)`**
   inserts into table Priviledge a record that assigns a developer whose id is developerId, the priviledge with priviledge name, to the page with pageId
3. **`deleteWebsitePriviledge(int developerId, int websiteId, String priviledge)`**
   deletes from table Priviledge a record that removes priviledge name from developerId, on websiteId
4. **`deletePagePriviledge(int developerId, int pageId, String priviledge)`**
   deletes from table priviledge a record that removes priviledge name from developerId, on pageId

When returning the priviledge instance, create a new priviledge instance and do not modify the existing priviledges instance provided in the parameters.

Exercise Your Data Model and DAOs (20pts.)

In a file called hw_jdbc_last_first.java, use the Data Model and Data Access Objects implemented earlier, to create the data shown below

1. (5pts.) Create the following developers and users. Insert into the correct tables depending on the type

id	Username	Password	First	Last	Type	Email	Key
12	alice	alice	Alice	Wonder	Developer	alice@wonder.com	4321rewq
23	bob	bob	Bob	Marley	Developer	bob@marley.com	5432trew
34	charlie	charlie	Charles	Garcia	Developer	chuch@garcia.com	6543ytre
45	dan	dan	Dan	Martin	Use	dan@martin.com	7654fda
56	ed	ed	Ed	Karaz	User	ed@kar.com	5678dfgh

2. (5pts.) Create the following web sites for the developers above. For both the created field and updated field, use the date your assignment will be graded, e.g., do not hardcode it

id	Name	Description	Owner	Editor	Admin	Visits
123	Facebook	an online social media and social networking service	alice	bob	charlie	1234234
234	Twitter	an online news and social networking service	bob	charlie	alice	4321543
345	Wikipedia	a free online encyclopedia	charlie	alice	bob	3456654
456	CNN	an American basic cable and satellite television news channel	alice	bob	charlie	6543345
567	CNET	an American media website that publishes reviews, news, articles, blogs, podcasts and videos on technology and consumer electronics	bob	charlie	alice	5433455
678	Gizmodo	a design, technology, science and science fiction website that also writes articles on politics	charlie	alice	bob	4322345

3. (5pts.) Create the following pages for the web sites above. Use the semester's start date for the created field. Use the assignment's due date for the updated field.

id	Name	Description	Website	Editor	Reviewer	Writer	Views
123	Home	Landing page	CNET	alice	bob	charlie	123434
234	About	Website description	Gizmodo	bob	charlie	alice	234545

345	Contact	Addresses, phones, and contact info	Wikipedia	charlie	alice	bob	345656
456	Preferences	Where users can configure their preferences	CNN	alice	bob	charlie	456776
567	Profile	Users can configure their personal information	CNET	bob	charlie	alice	567878

4. (5pts.) Create the following widgets for the pages shown.

Name	Type	Text	Order	Width/Height	Url	Page
head123	heading	Welcome	0	null	null	Home
post234	html	Lorem	0	null	null	About
head345	heading	Hi	1	null	null	Contact
intro456	html	Hi	2	null	null	Contact
image345	image	null	3	50x100	/img/567.png	Contact
video456	youtube	null	0	400x300	https://youtu.be/h67VX51QXiQ	Preferences

Implement Updates (15pts.)

Using the DAOs implemented earlier, do the following updates

1. Update developer - Update Charlie's primary phone number to 333-444-5555
2. Update widget - Update the relative order of widget head345 on the page so that it's new order is 3. Note that the other widget's order needs to update as well
3. Update page - Append 'CNET - ' to the beginning of all CNET's page titles
4. Update roles - Swap Charlie's and Bob's role in CNET's Home page

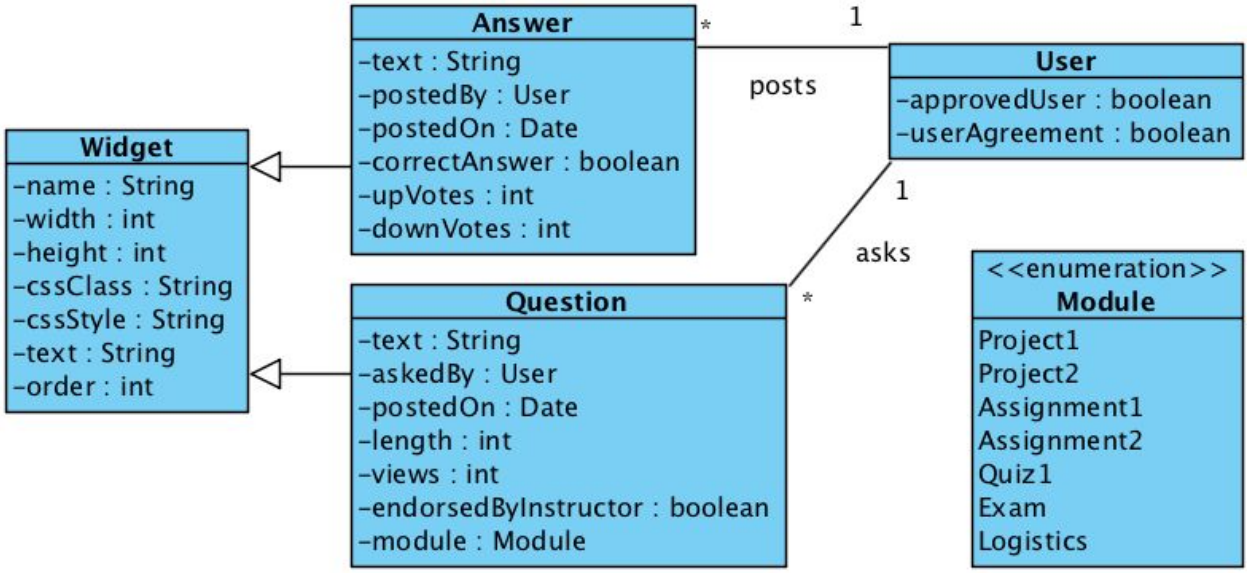
Implement Deletes (15pts.)

Using the DAOs implemented earlier, do the following deletes

1. Delete developer - Delete Alice's primary address
2. Delete widget - Remove the last widget in the Contact page. The last widget is the one with the highest value in the order field
3. Delete page - Remove the last updated page in Wikipedia
4. Delete website - Remove the CNET web site, as well as all related roles and privileges relating developers to the Website and Pages

Stored Procedures (10pts.)

Consider the following class diagram



Create stored procedures for the following cases.

1. Create a Stored Procedure called ***getUnansweredQuestions*** which retrieves the question for each module with most number of answers and no correct answer. Group the questions by module and print the question text and numbers of answer for the question. (5M)
2. Create a Stored Procedure called ***endorsedUsersForWeek***, which retrieves the 5 most endorsed users (having the most number of correct answers for the questions posted in the given week). If there multiple endorsed Users per week, sort by their First Name. (5M)
   { The stored procedure will be given two date values representing the start and end of a week. (These date values will be in the form java.sql.date) Based on the date values, filter the users and print to the console the userID, user full name, endorsedUser. }

Deliverables

In the folder provided to you, submit your entire code into the directory on Github. And also make a submission on Blackboard with the submission text as your Github URL.

FAQ

Q: I made the assumption “only primary phone numbers and address can be updated or deleted”, because it atleast solves the purpose of the implement updates and deletes part.

A: There are quite a few ways to do this depending on how general you want to get. Here we can assume we can be very specific to the use case of updating, deleting or adding a phone or an address to comply with the very narrow set of use cases. A more general solution would keep track of the changes to the objects in memory and update the database based on only those fields that changed. That’s what JPA does. It keeps track of the lifecycle of a managed object and updates the database accordingly. For instance setter functions could keep track whether a particular field is dirty and should be included in the eventual update statement. A remove method in Java would flag the object being removed for eventual

removal from the database. But this is far too ambitious and it’s not the intention of this assignment to rewrite JPA. Instead it is intended to introduce you with the challenges of mapping a flat relational model to a hierarchical transient data model.

Q: "retrieves the 5 most endorsed users (having the most number of correct answers for the questions posted in the given week)." has two different ways to interpret:
1) the users that have asked the most questions that have the endorsedByInstructor field set to true (implied by "most endorsed users")
2) or the users that have the most answers with the correctAnswer field set to true (implied by "the most number of correct answers")
And if it's option B, should we be using the date that the question was posted at, or the answer was posted at?

A: Use the option B and the date the question was posted

Q: My question is that we are asked in the assignment to create a constructor that can create an instance of Widget class. But from my understanding the Widget cannot exist as-is. It has to be one of Image, Youtube, Html or Heading. And following that design pattern my Widget class is abstract. Just like Person class is abstract and Developer and User are concrete classes that implement Person.

A: There might never be a need to actually instantiate a Widget per se. A widget instance means nothing outside the context of the derived classes. In that sense you could consider it an abstract class, e.g., never actually instantiated, and perhaps you should try making abstract. The usefulness of breaking the datamodel into several classes that inherit from each other is more as a convenience to group together properties that are common to several subclasses. The constructors in Widget are there as a convenient way to break up the chore of initializing and validating the constructor parameters. Derived constructors would delegate to super constructors those fields that are common to sibling classes in the inheritance hierarchy.