Q1. NOUNS:

**Customer**,

**DevAccount**, username, password,

**Developer**, fName, LName, dob, email, website links, billing address, primary address, other address, primary phone, other phone,

**Address**, street1, street2, city, state, zipcode,

**Phone**, country code, area code, phone number,

**Owner**,

**Website**, website name, website description,

**Website Metadata**, date created, last updated, owner, visit, stylesheet,

**Role**, Admin, Editor, Reviewer,

**Website Privilege**, Create, Read, Update, Delete,

**Webpage**, page name, page title, page description,

**Widget**, id, name, top, bottom, left, right, width, height, foreground color, background color, css class, css style

**Heading Widget**, Text Size, one, two, three, four, five, six

**Image Widget**, url, x position, y position,

**Horizontal Align**, left, center, right,

**Vertical Align**, top, middle, bottom,

**Events**,

**Page Events**, onLoad, onUnload

**Widget Events**, onChange, onClick, onDrag, onEnter, onExit

Q2 & Q7. VERBS and CARDINALITY:

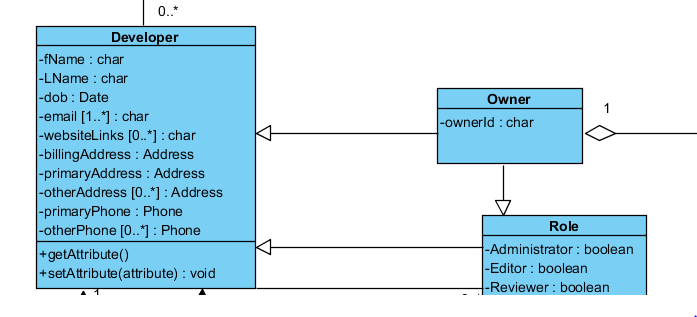
All associations, aggregations and compositions are weak-weak many-to-one or one-to-many relations in his class diagram except for Generalization where there is one to one correspondence due to Foreign Key also being Primary Key.

1. Customer(many) *registers* for a DevAccount(one)
2. Developer(many) *logs in* to his DevAccount(one)
3. Developer(one) *configures* Address(many)
4. Developer(one) *configures* Phone number(many)
5. Owner(one) *is a* Developer(one)
6. Owner(one) *is a* Role(one)
7. Role(one) *is a* Developer(role) (one)
8. Developer(one) assigns/revokes role(many)
9. Owner(one) *creates* the website(many)
10. A Role (one) *gets* website privileges (zero or one) to *edit the website* (one) or *add a webpage* (one) depending on what kind of privileges he has been given
11. A Website (one) *reveals* *details* when selected and shows website metadata (many)
12. A Webpage (one) *reveals* *details*  when selected and shows webpage metadata (many)
13. Website (one) *has* Webpages (many)
14. Webpage (one) *has* Widgets (many)
15. Webpage (one) *associated to* Events (many)
16. Widget (one) *associated to* Events (many)
17. PageEvent (one) *is an*  Event (one)
18. WidgetEvent (one) *is an*  Event (one)
19. HeadingWidget(one) *is a* Widget(one)
20. ImageWidget(one) *is a* Widget(one)
21. YoutubeWidget(one) *is a* Widget(one)
22. ParagraphWidget(one) *is a* Widget(one)

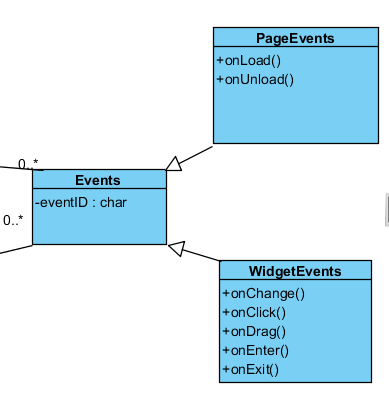
Q3. Generalization/ Specialization:

This class diagram has the following generalizations-

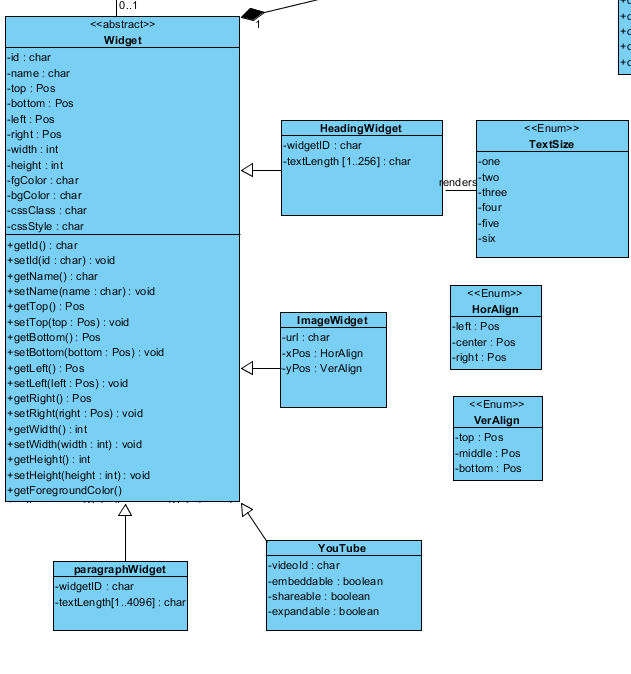
1. Owner *is a* Developer: As mentioned in the problem statement, the very first developer that originally created the Website is being called Owner. Therefore the Owner too is a Developer.
2. Owner *is a* Role: This inheritance holds since Owner is one of the “roles” that a Developer can play
3. Role *is a* Developer: Since all roles are being played by only the Developer, in essence any kind of role is a Developer role.



1. PageEvent/ WidgetEvent *is an* Event: Events are of two types: PageEvents which run on pages and WidgetEvents which run on Widgets. Each type of Event has their own set of methods/operations.

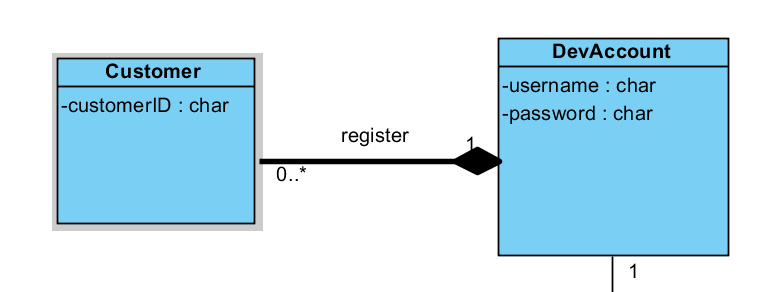


1. Heading/ Paragraph/YouTube/Image *are* Widgets: Each is a special kind of Widget that share common Widget attributes

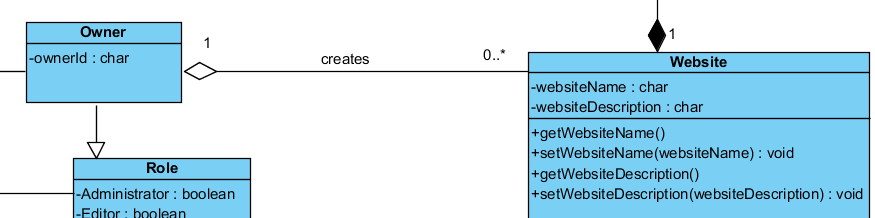


Q4. Aggregation and/or Composition:

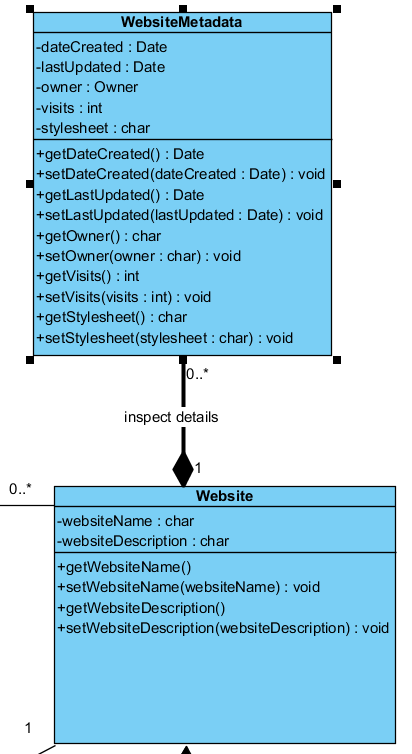
1. DevAccount *is composed of*  Customers: If a customer’s Developer Account is deleted he/she is no longer a Customer to the Firm.



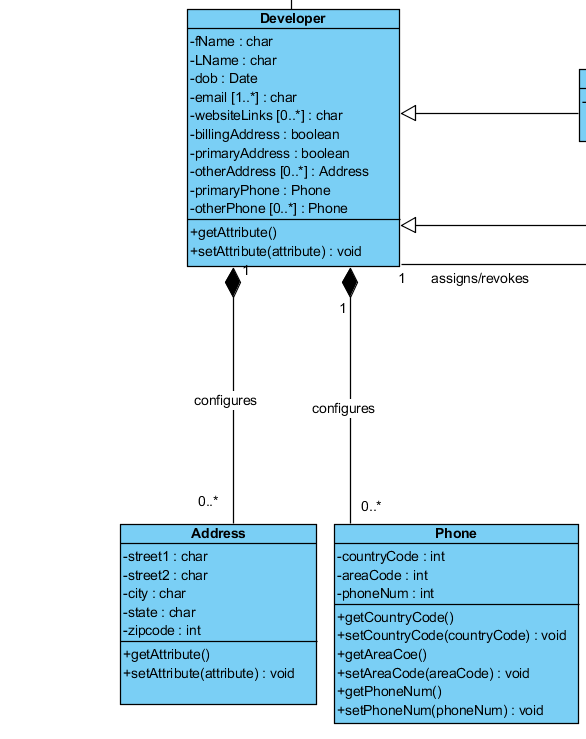
1. Website *is a part of* Owner: Since Owner originally creates the website, it belongs to him or is a part of him. However, if the owner ceases to exist in the database, the business logic dictates that we do not want his Website to be removed from the database as well. Hence the existence of the Website is independent of the existence of the owner even though it is a part of the Owner. Hence this is an aggregation



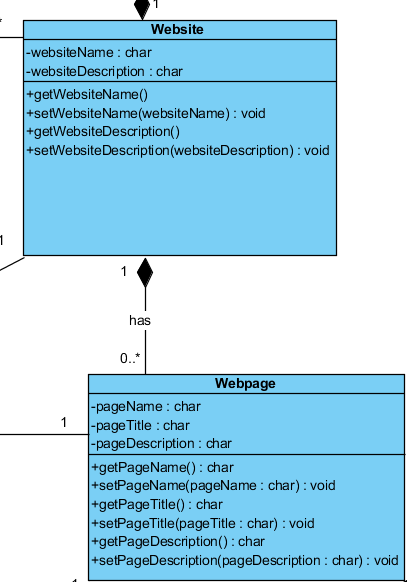
1. Website is composed of Website Metadata: When a Developer selects a website, additional details about the Website i.e its metadata is revealed. Since the metadata is core information about the website and cannot exist without the other hence this is a compostion.



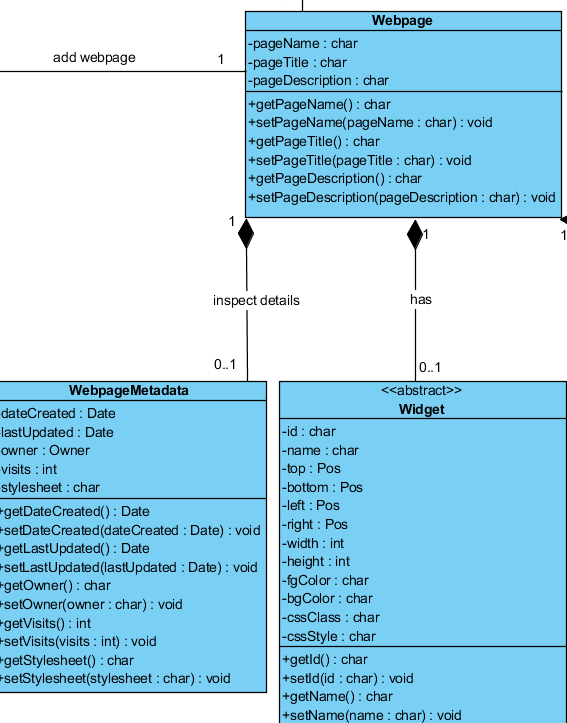
1. Developer *is composed of* Address and Phone: Developer configures his Address and Phone information. However if a Developer ceases to exist there is no point in still keeping a record of his Address and Phone details in the database. Infact also for security reasons it is actually a good idea to cascade delete all records of such a Developer. Hence this is a compostion



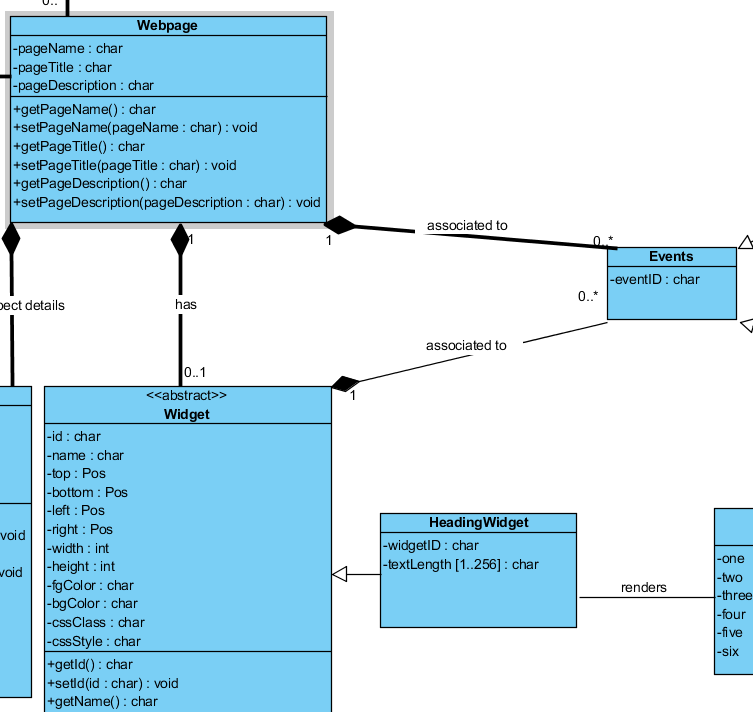
1. Website *is composed of* Webpages: This is pretty self-explanatory. If a Website is removed, so will all its webpages, no point in keeping orphaned webpages in the database.



1. Webpage *is composed of* Widgets: No point in having orphaned widgets if there is no webpage to display them on.
2. Webpage *is composed of* Webpage Metadata: Similar to website being composed of website metadata.



1. Webpage/ Widget is composed of Events: As mentioned earlier in Q3 pt.4, each webpage and widget has associated with it scripts that handle webpage events for webpages and widget events for widgets respectively. These are together generalized as a class of events and a webpage or a widget is composed of these events. If the webpage or the widget is deleted, all events running on it will also have to be removed since the agent being acted upon itself is removed.



Q5. Classes vs Attribute analysis:

Classes and Attributes are the UML or conceptual model equivalent of Tables and columns in a Relational Schema.

1. Often times an attribute may be important or have the characteristics of differentiating itself that it deserves its own class. For example the Owner class could have been just an attribute of the Role class but it has special characteristic that only an Owner can create the original website and not any other Developer Role. Hence Owner got its own class.
2. Instead of putting username and password as attributes in the Developer class, I distributed those attributes into a separate DevAccount class to reify the many-to-many Customer-Developer relationship.
3. Instead of listing all event based methods in the same Event class its better to distribute them into separate child-classes Webpage Events and Widget Events and only keep the common attributes of either kind of event in the Event class.
4. Similarly attributes of Widgets class has been distributed into the various kinds of its child Widget classes.
5. Address and Phone are plural with more specific attributes. Had the granularity of the scope of requirements not been so detailed, both classes could have been present as just attributes in the Developer class. But because of more granularity in requirement we had to separate them out into separate classes.

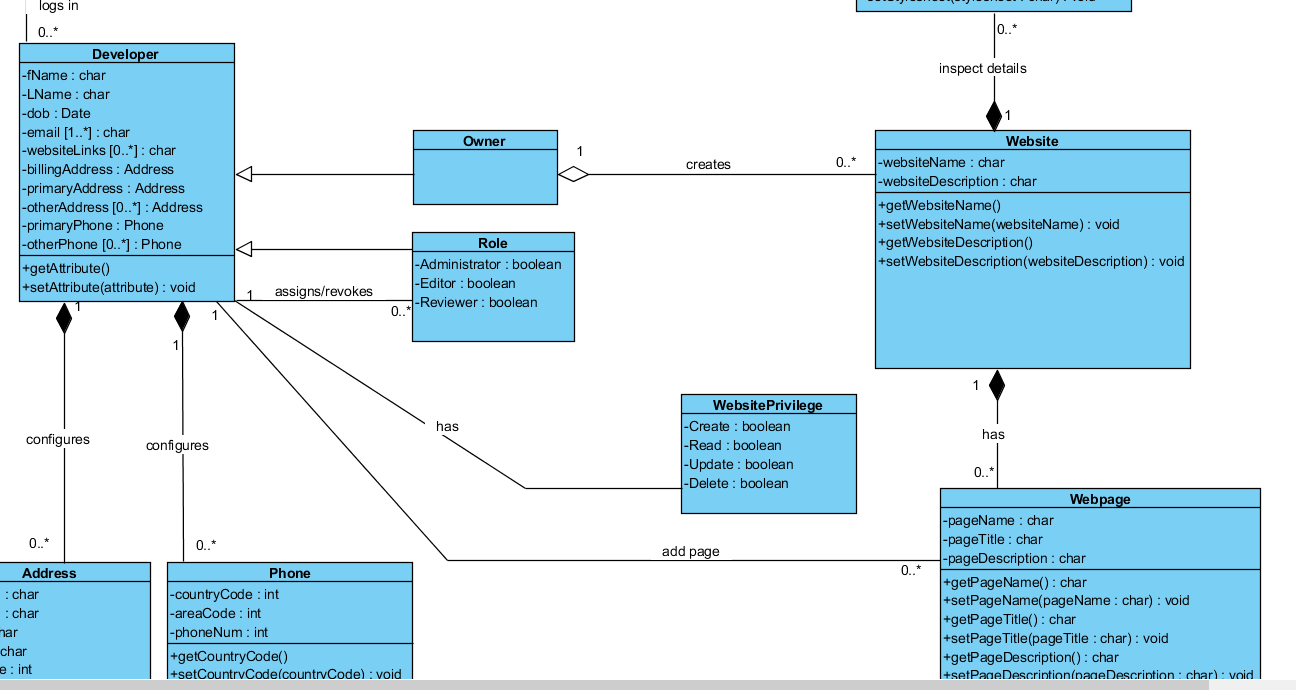
Q6. Correct Data Types, e.g., Date, String, Integer, List, Array, Enumeration, etc.

1. Date: dob, metadata storing Date of creation and Date of last update on a website or webpage
2. String: Any literal value, names, ids, urls, addresses, usernames, passwords, Stylesheets etc. (Note in my diagram wherever I have mentioned “char” , it is actually “String” since the UML drawing tool I was using- Visual Paradigm gave the default option as char. This is true except two instances- textLength attribute in HeadingWidget and ParagraphWidget)
3. Integer: zipcode, phone details, number of visits to a website or a page, widget dimensions- width, height, widget position
4. List/Array : textLength attribute in HeadingWidget and ParagraphWidget
5. Enumeration : TextSize, HorAlign ( Horizontal alignment), VerAlign (Vertical alignment)

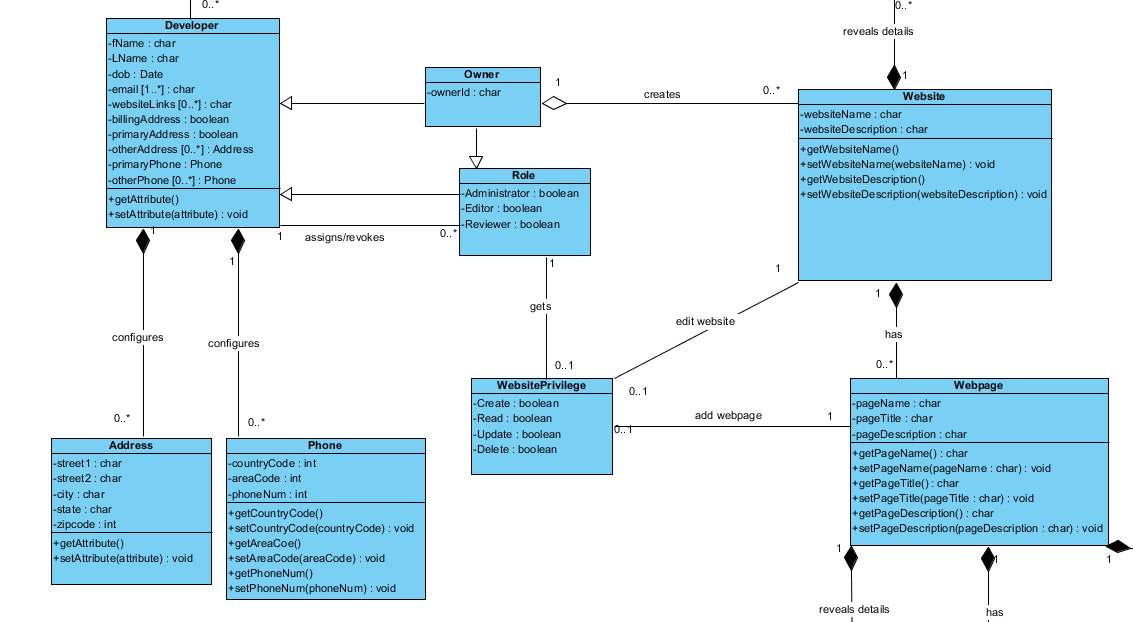
Q8 & 9. Removing inadequate relationships and Reifying

1. In the non-reified preliminary version of the class diagram we **oversimplified** and said :
2. Developer has WebSitePrivileges
3. Developer adds WebPages

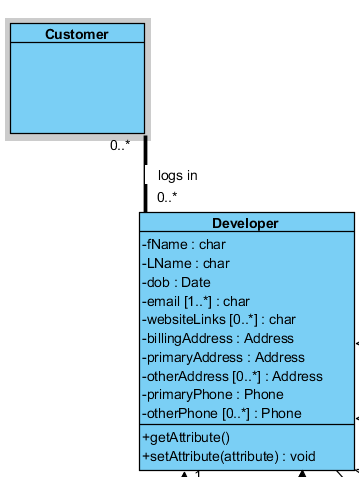
However there was no restriction for any Developer to add Webpages regardless of their privileges and whether the were granted those privileges or not.



In actuality this is a 3-way relation and each relation takes part only if certain privileges were granted or they don’t participate at all. This is captured by reifying by adding a new mapping via WebSitePrivileges table:



1. Another reification done was on the weak-weak Customer-Developer relationship.



To fix this we used a new mapping table DevAccount that created two new Many-to-one and one-to-many relations.

