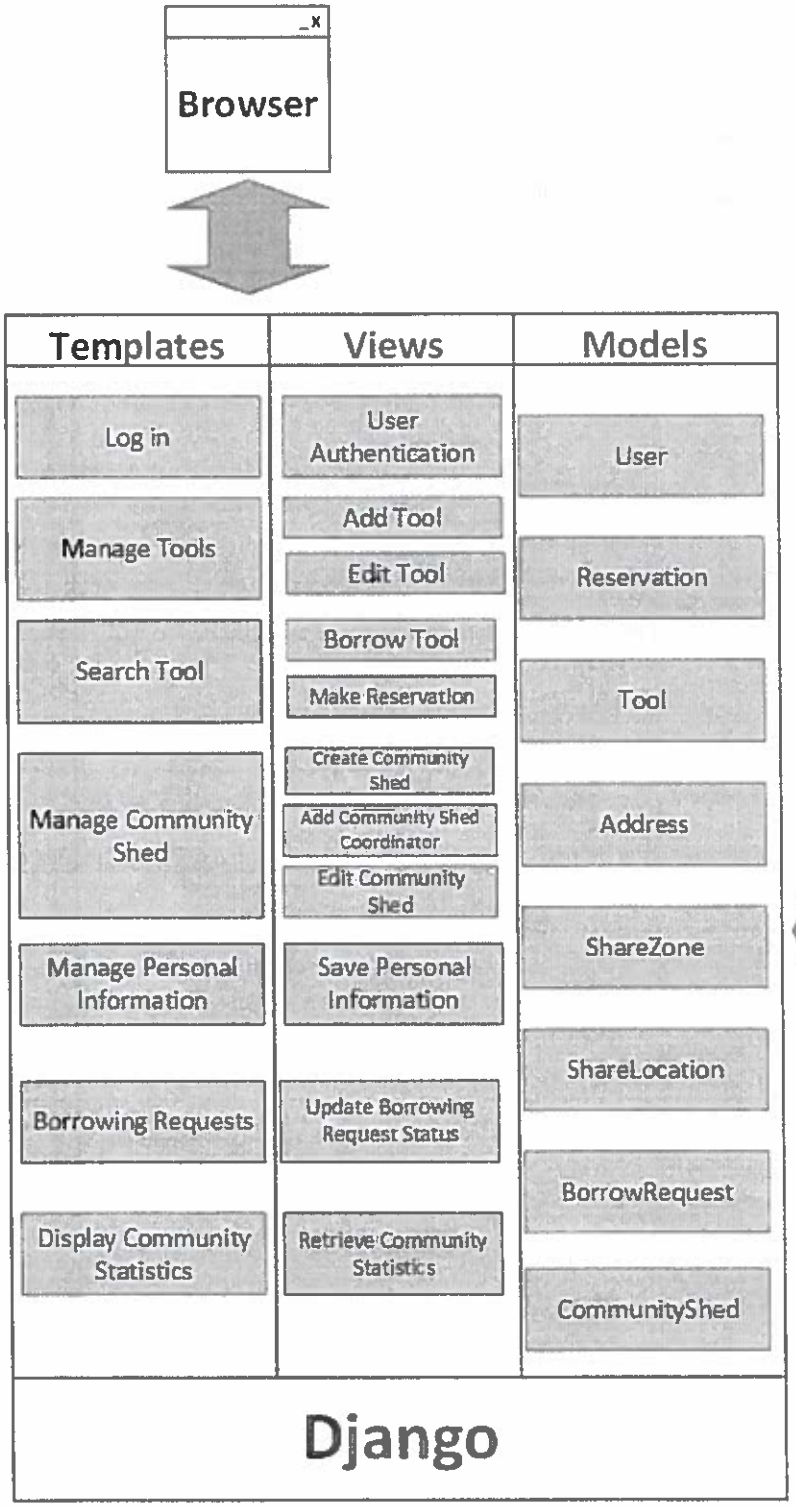
Product Design

Team f610-01f tigerPaws

Architectural Overview



- Quite a few grammatical errors. Use spellcheck.

Prototype: (working)
- plain html page

Good. Well done at things. 8.5/10

Components and Functions

<Django User>, <ToolUser>	<Component state • User maintains the personal information of user. • What services does the component provide to other components? • User provides source of information for other components such as Share Zone, Tool, and Address. • Components such as Reservation and Community shed depends on User <Component behavior • getTools() get all the list of tools.
<AuthUser>	<Component behavior • isAuthenticated() will check the authentication od the user.
<Address>	<Component state • It stores the address of the User. It consist of address1, address2, city, state, zipcode and country. <Component behavior • This address can also be the address to share the tool depending upon the user. • It is used by <User> for ?
<Tool>	<Component state • The Tool consist of detailed information of tool such as id, description, image, owner, and status, location of the tool from where it can be shared and pickup arrangements. <Component behavior • Share() will implement functionality of sharing among users. • Borrow() implies functionality of borrowing tool. • isInCommunityShed returns Boolean value to check whether tool's availability. • canBeBorrowed returns Boolean value wheather toolcan be borrowed.
<Community Shed>	<Component state • Each Community maintain information about coordinator who is the owner of Shed and its address. <Component behavior • Community Shed is one of the places from where the tool can be stored physically and shared.User can register its tool and can store it in Community Shed of his Share Zone. Community Shed have list of all tools registered by users in same share zone and which they share it by Community Shed other than home. • It implements functionality of adding tools, removing tools and getting list of tools.
<Share Zone>	<Component state • Share Zone is determined by the zip code of your address. It is unique to every user. • Zip code is the primary identifier to determine the Share Zone of the user. • Share Zone have the list of all tools that people in that area wants to share. So it is easy to get the tools available for sharing <Component behavior • getTools() will fetch all the tools in share zone.
<Reservation>	<Component state • It consist of information such as ID, Borrower, Tool, Start Date and End Date. • It depends on Tool, such that if the tool is available for sharing it can initiate the process to reserve tool. The borrower attribute is the User component.
<BorrowRequest>	<Component state • It includes information such as request message ,tool to borrow, Start Date ,End date, status, • This depends on User and Tool to fetch information regarding tool and user, When the user sends borrow request it can check it status. <Component behavior • Methods such as approve () and reject () approves or reject the tools.
<ToolStatus>	<Component state • It shows the status of wheatear it is available or no. • This is used by Tool component to show the status of tool.— behavior

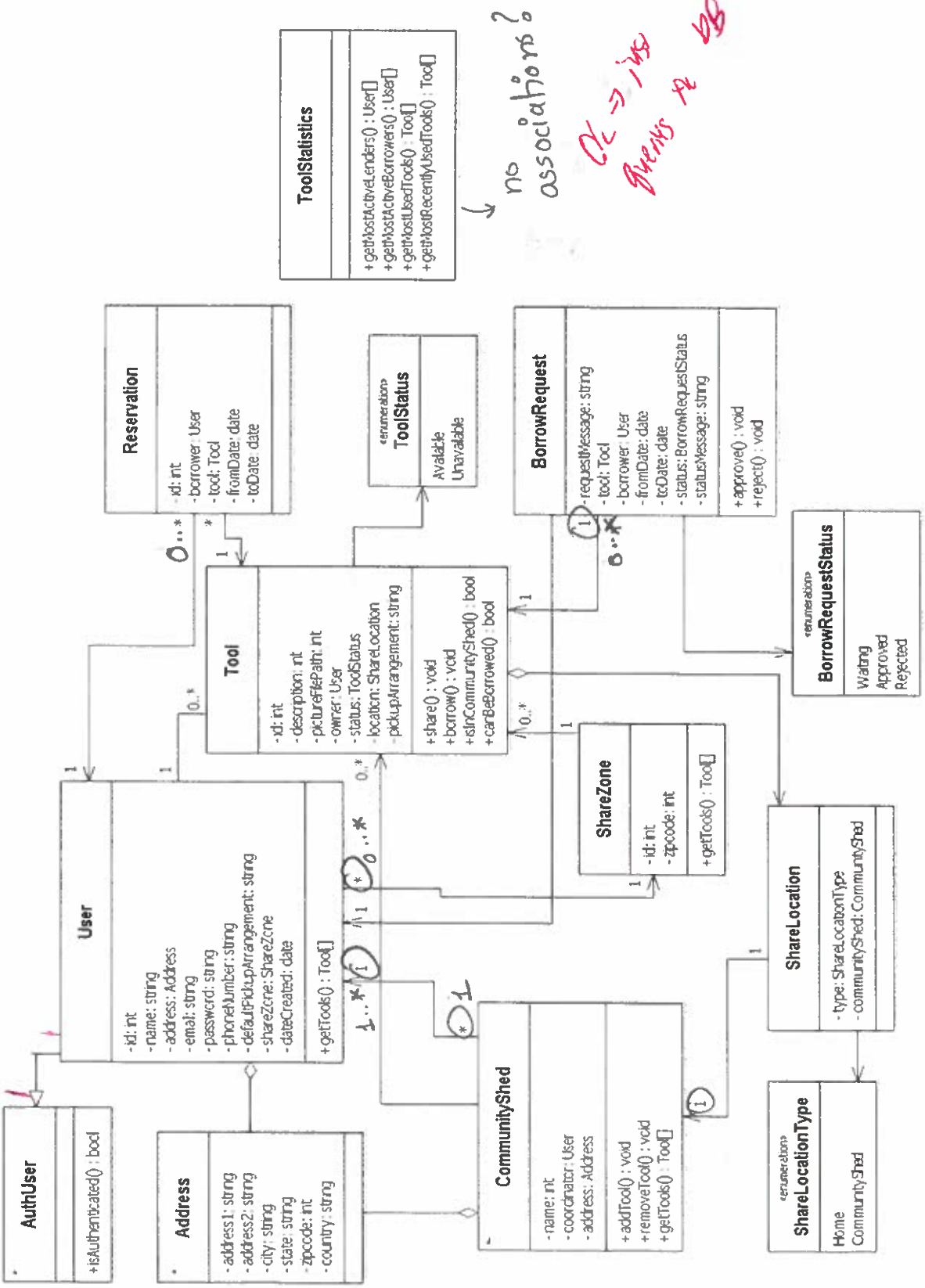
<BorrowRequestStarus>	<Component state • It displays the status of the Borrow Request initiated by the user. • It has three status Waiting, Approved, rejected.
<ShareLocation>	<Component state • It is associated with Community Shed to determine the location from where the tool can be shared. • It also shows to which community Shed it is linked with. <Component behavior • It depends on Community Shed to display the information.
<ShareLocationType>	<Component state • This component is derived from Share Location component . • It have two attributes Home and Community Shed. <Component behavior • It determines the location from where the tool can be shared.
<ToolStatistics>	<Component state • It returns back the statistics of tools. <Component behavior • getMostActiveLenders() • getMotActiveBorrowers() • getMostUsedTools() • getMostRecentlyusedTools()

For component behavior, it is not necessary to explicitly mention the function name. You only need to mention the high level functionality

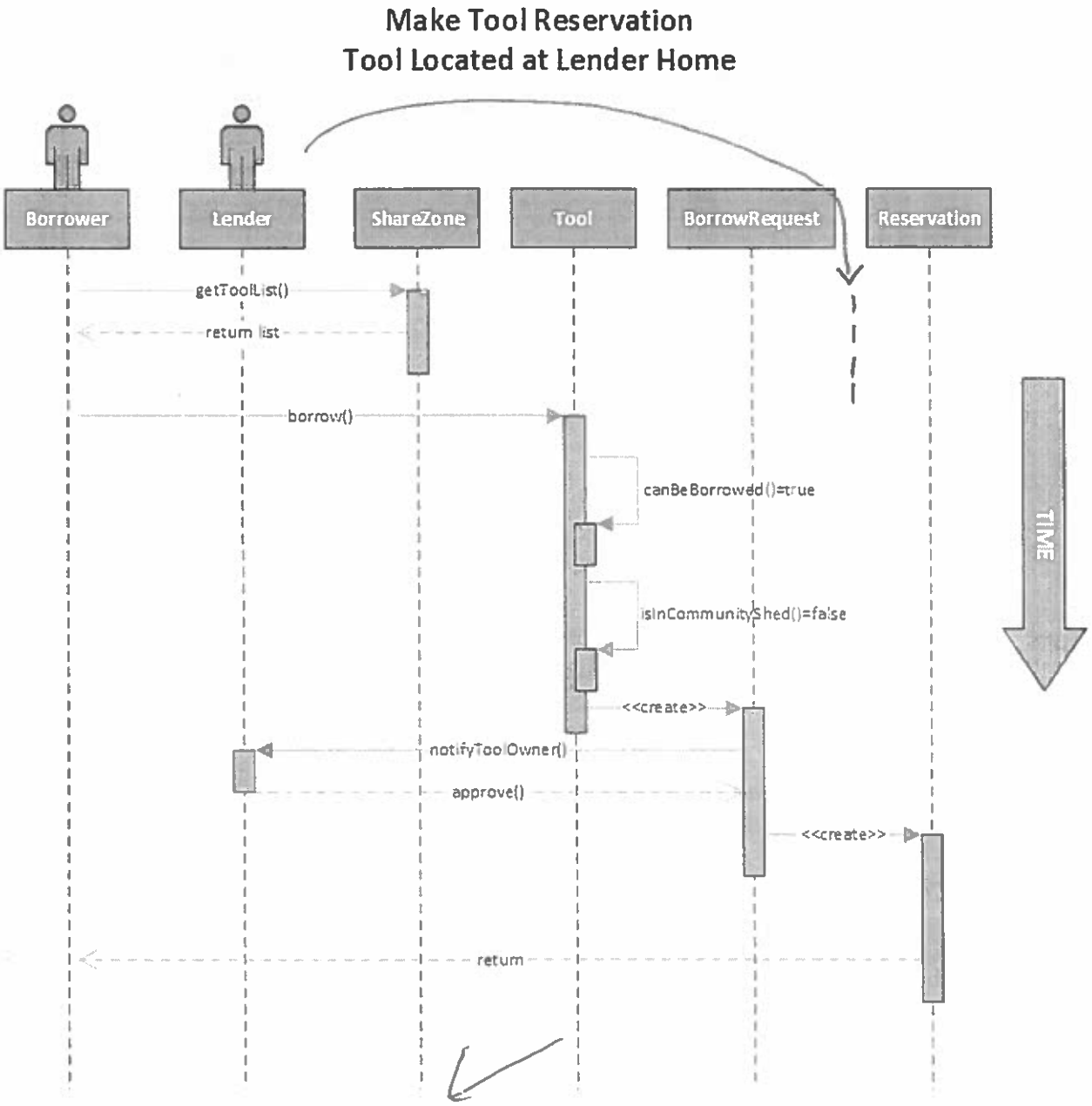
Eg. Tool Statistics :

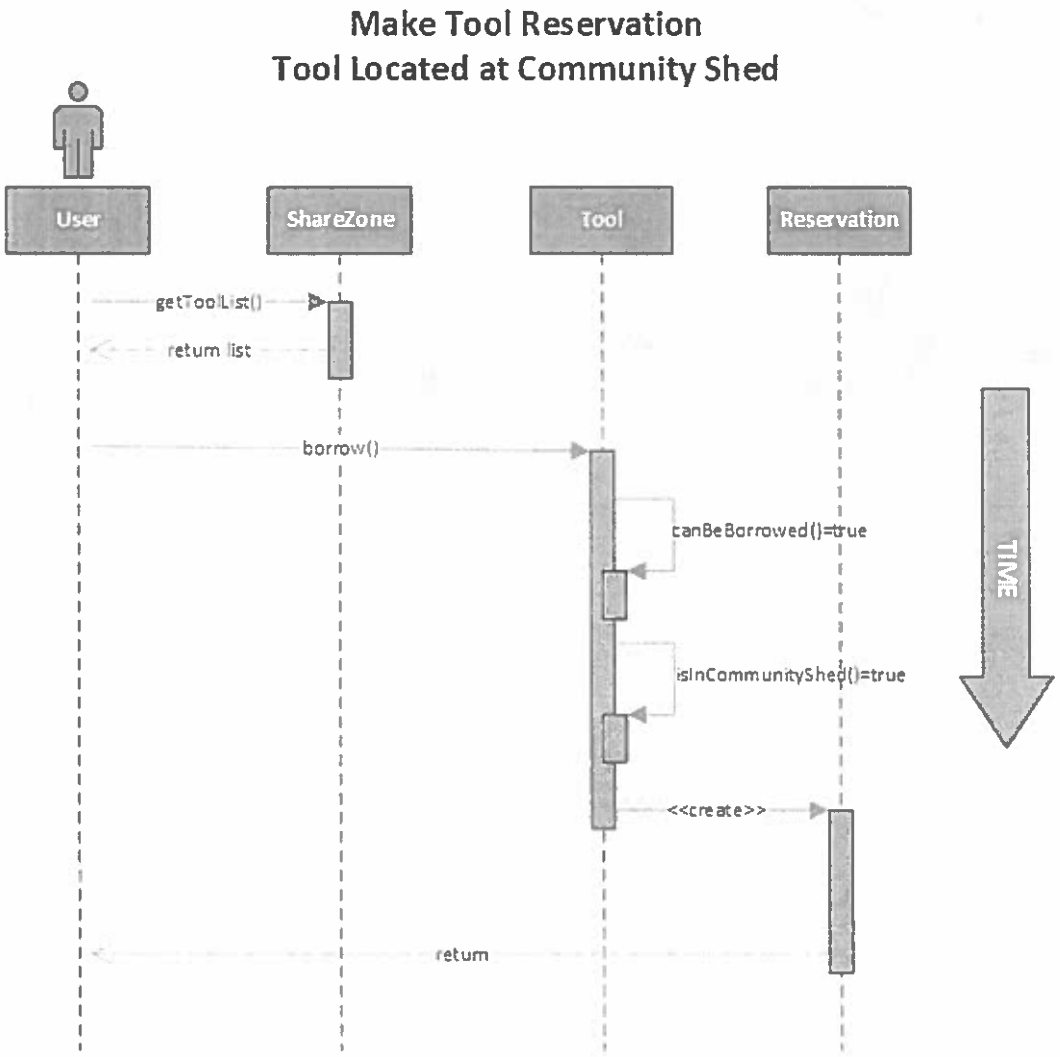
Comp behavior - This component provides statistics on the users in a share zone such as the most active lender & borrower . . .

Class Diagram



Sequence Diagrams





Design Rationale

The first design decision that resulted in considerable debate was whether the architecture needed a superclass for the Location of a tool, which could then be sub-classed into Home and CommunityShed locations. Each of these children would inherit a physical address and functionality from the parent class. The CommunityShed class would then have the class-specific attribute of Coordinator while Home would have the class-specific attribute of Owner. We eventually decided not to pursue this line of architecture, as it would violate the "Don't Repeat Yourself" (DRY) principle of software engineering. The Tool already has an Owner attribute, which uses ToolUser as a foreign key inside the SQLite database. Tools can also easily get a Home location from the address of its Owner. Having this redundancy could cause us implementation problems in the near future, as it is a duplication of data that would need to be updated every time the user changed his or her personal information. Instead, we decided to implement a ShareLocation enumeration that allows the Tool itself to query where it is being shared from. As shown in the sequence diagrams, this allows the tool to control the initial flow of the reservation procedure, by first checking whether it can be borrowed and then checking if it is in the CommunityShed. Different answers to these two queries will result in alternate flows through the application. We also made the decision to use this class called ShareLocation to indicate whether a tool is shared from Home or from the Community Shed rather than a different

type of implementation. This decision improved modularity and encapsulation. We further chose to use a CommunityShed class to encapsulate its functionality and allow the Coordinator to manage its tools from methods contained within the class, increasing encapsulation.

The second major design consideration was how to use Django's authentication system and inherent User class to our advantage. Originally, the team had created its own User class with fields like username, password, first name, and last name. Using Django's class allowed the framework to handle these attributes for us. However, there was discussion over how to best incorporate the extra attributes we wanted a ToolUser to contain. One way was to have the ToolUser class contain a Django User, and use that containment relationship to access the Django User's variables. However, we decided to subclass the Django User to make ToolUser a child class, as a better way to implement an object-oriented design of that relationship. That way, the ToolUser has all the functionality and methods of the parent, but can add application-specific functionality and data that our project requires. This seemed to be the correct way to design the system, but may be something we need to readdress as issues arise during development.

In the ShareLocation class, it contains a CommunityShed attribute to facilitate the process of retrieving the CommunityShed object that contains certain tools in its zip code. This might add some redundancy to the system and is something we will monitor carefully as development continues. With redundancy like this, all changes to the CommunityShed field must be made using methods on the underlying object so that the data is changed in only one location, thereby ensuring the continued consistency of the system.

Pickup Arrangement was declared as a string instead of as a class since it will just hold a description about how a certain tool should be picked up by the Borrower, rather than contain any specific functionality or other data types. Thus, the encapsulation and abstraction of a class seemed to be more than was really required for this particular implementation requirement (we decided not to use a hammer when a fly-swatter would do).

Address is a class instead of implementing it as a string because the zip code attribute is used to determine the ShareZone to which a specific address belongs. Thus, the address can be queried with methods from other classes besides the User to retrieve this critical information. We also decided to make ShareZone a class in order to facilitate retrieving tools that belong in a given ShareZone. We decided this was better than delegating this responsibility to the Address class because this separates functionality into logical containers and also increases encapsulation. This type of separation of concerns makes it easier to adjust functionality later on in the development process if requirements change. For instance, we could adjust a ShareZone to allow for multiple zip codes or allow a tool to be put into multiple ShareZones. These types of adjustments would not be possible inside the Address class, and this decision further increases encapsulation. We decided to hide from Address that information that it does not need to know about to execute its responsibilities.