Exercise 1:

1. What are two invariants of the state? (Hint: one is about aggregation/counts of items, and one relates requests and purchases). Say which one is more important and why; identify the action whose design is most affected by it, and say how it preserves it.

The first invariant is that the counts of items requested must be non-negative. The second is that a purchase cannot exist without a matching request. The more important invariant is the one that relates requests and purchases, as something should not be bought that is not specifically requested by the user. The counts being non-negative is important, but is not vital, since a higher count of the item being bought is not a major problem. The purchase action is most affected by this invariant, and it preserves it by requiring a request to exist for the item.

2. Fixing an action. Can you identify an action that potentially breaks this important invariant, and say how this might happen? How might this problem be fixed?

The removeItem action might break this invariant because if there are purchases associated with an item and that item is removed, purchases will exist without a matching request. This problem could be fixed by either requiring the item not to have any associated purchases before removal or automatically removing all purchases associated with an item.

3. The operational principle describes the typical scenario in which the registry is opened and eventually closed. But a concept specification often allows other scenarios. By reading the specs of the concept actions, say whether a registry can be opened and closed repeatedly. What is a reason to allow this?

A registry is allowed to be open and closed repeatedly. This could exist to allow a user to close the registry while they add items and open it again when they're ready.

4. There is no action to delete a registry. Would this matter in practice?

This would not matter in practice to the user because closing a registry is effectively deleting it for everyone but the user. At a larger scale, though, this action might need to be implemented to avoid accumulating unnecessary data.

5. What are two common queries likely to be executed against the concept state? (Hint: one is executed by a registry owner, and one by a giver of a gift.)

The common query to be made by the registry owner will be the set of purchases to see what others have been buying. For the givers, this query will be the set of requests while they figure out what to purchase.

6. A common feature of gift registries is to allow the recipient to choose not to see purchases so that an element of surprise is retained. How would you augment the concept specification to support this?

An additional flag on the registry object that indicates whether or not that user prefers not to see the purchases. If this flag is active, then when the user queries the purchases, they will be blocked from seeing the counts.

7. The User and Item types are specified as generic parameters. The Item type might be populated by SKU codes, for example. Explain why this is preferable to representing items with their names, descriptions, prices, etc.

This representation is preferable because it is a unique identifier, whereas an identifier like a name could be vague depending on the item.

Exercise 2:

1. Complete the definition of the concept state.
2. Write a requires/effects specification for each of the two actions. (Hints: The register action creates and returns a new user. The authenticate action is primarily a guard, and doesn't mutate the state.)

concept PasswordAuthentication
purpose limit access to known users
principle after a user registers with a username and a password,
  they can authenticate with that same username and password
  and be treated each time as the same user
state
  a set of Users with
    a username
    a password
actions
  register (username: String, password: String): (user: User)
   requires username to be unique and password to be non-empty
   effect creates a user object with the given username and password and returns it
  authenticate (username: String, password: String): (user: User)
   requires username and password pair to exist in the set of users
   effect allows access to the data of that given user

3. What essential invariant must hold on the state? How is it preserved?

The essential invariant is that a user can only access data once they are authenticated. This is preserved by checking for matching username and password pairs before allowing authentication.

4. One widely used extension of this concept requires that registration be confirmed by email. Extend the concept to include this functionality. (Hints: you should add (1) an extra result variable to the register action that returns a secret token that (via a sync) will be emailed to the user; (2) a new confirm action that takes a username and a secret token and completes the registration; (3) whatever additional state is needed to support this behavior.)

concept PasswordAuthenticationWithEmail
purpose limit access to known users
principle after a user registers with a username and a password,
  they must be confirmed by email and after that
  they can authenticate with that same username and password
  and be treated each time as the same user
state
  a set of Users with
    a username
    a password
    an email
    a secretToken
    an active flag
actions
  register (username: String, password: String, email: String): (user: User, secretToken: string)
   requires username to be unique and password to be non-empty
   effect creates an inactive user object with the given username and password and emails the secretToken to the user
  confirm (username: String, secretToken: String): (user: User)
   requires secretToken to be associated with the user
   effect completes registration of the user and flags them as active
  authenticate (username: String, password: String): (user: User)
   requires username and password pair to exist in the set of users
   effect allows access to the data of that given user

Exercise 3:

concept PersonalAccessToken
purpose an alternative to passwords for authentications in specific cases
principle a currently registered user can request a token to use
  this random token will be created with an expiration date
  after this, the token can be used to access the specific scopes set
state
  a set of Users with
    a username
    a password
    a token

a token with
    an expiration date
actions
  createToken (tokenName: String, expiration: String): (token: string)
    requires expiration to be a valid date
    effect creates a token that is ready to be used
  authenticate (username: String, token: String): (user: User)
    requires token to be associated with that user
    effect allows access to the data of that given user within the initial scope

These concepts differ mostly in terms of simplicity. The token is created once the user is already authenticated and is meant to be used as a key that can be saved instead of having to remember a password. Although it is easier to use, the token might be restricted in scope and does not have as many capabilities as a user authenticated with a password.

Exercise 4:

concept ShortURLs
purpose a method of sharing URLs that are not as long as typical ones
principle the user may enter a URL
    this URL will then be converted into an autogenerated shorter version
    the user then has the option to change the URL into a user-defined version
state
  a set of ShortURLs with
    an original URL
    an expiration date
actions
  createURL (originalURL: String): (result: shortURL)
    requires originalURL to be a valid URL
    effect creates an autogenerated shortened version of that URL with an automatic expiration date
  customizeURL (originalURL: shortURL, newURL: String): (result: shortURL)
    requires originalURL to be in set of shortURLs and newURL to be unique
    effect changes the originalURL to match the newURL text

concept BillableHoursTracking
purpose a tool used to track work by the hour
principle an employee is able to start a sessions describing work to be done
that employee is then able to end the session and alter it if needed
state
a set of sessions with
a start time
job description
an automatic expiration date
a set of completedSessions with
a start time
an end time
job description
a total number of hours worked
actions
startSession (startTime: Time, description: String): (result: session)
requires startTime to be the current time or later
effect creates a session with the inputted description and start time. Auto-generates an expiration date 12 hours after the start time, which will automatically end the session if it is not manually ended
endSession (sessionToEnd: session): (result: completedSession)
requires sessionToEnd to be an active session
effect ends the input session and adds it to the set of completed sessions
alterSession (sessionToAlter: completedSession, newEndTime: Time): (result: completedSession)
requires sessionToAlter to be in set of completedSessions
effect changes the input session end time to the input time and updated the total number of hours worked (used in case employee forgets to end session)

concept ConferenceRoomBooking
purpose a tool used to book conference rooms
principle a user is able to search a specific time and see what rooms are available
the user is then able to select from available rooms to book for a certain time
state
a set of rooms with
a set of reservations
a room number
an availability time
a set of reservations with
a user name
a date
an associated room
a start time
an end time
actions
searchTimes (date: Date, startTime: Time, length: Number): (result: Rooms)
requires startTime + length to be during available hours on the input date
effect results in a set of rooms available during the given time and date
reserveRoom (user: String, room: room, date: Date, startTime: Time, length: Number): (result: reservation)
requires room to be available during startTime + length
effect creates a reservation for the input user on the given date, times, and room, and adds that reservation to the set