### **Task 2:**

**Finding the Responsible Class for a Method:**

Apply the **Information Expert Principle**. This principle suggests assigning a responsibility (which might be implemented as a method or operation in a class) to the class that has the information required to fulfill it. To do this, first clearly state the responsibility, then identify which classes have the necessary information. This often leads to a design where a class handling a concept in the real world is also responsible for operations related to the data (attributes) it holds.

**Finding the Responsible Class for Creating a New Object:**

Use the **Creator Principle**. A class B should be given the responsibility of creating instances of class A if:

* B aggregates A objects (meaning A objects are part of a larger whole, B).
* B contains A objects.
* B records instances of A objects.
* B closely uses A objects.
* B has the initial data needed to create A objects.

When there are multiple options, prefer assigning the creation responsibility to a class that aggregates or contains the objects. The main idea is that the creating class will likely need to have an association (a link) to the created object anyway and will use it frequently.

**Task 3:**

**Class: User**

* **Responsibilities:** Knows User **Attributes** (id, name, contactinfo). Can start the process to register and manage subscriptions (acting as a local Controller for these actions).
* **Collaborators:** Subscription (manages associations to Subscriptions).

**Class: Subscription**

* **Responsibilities:** Knows Subscription **Attributes** (subscriptionId, frequency). Knows its associated User and Website (Information Expert about itself). Can update its own settings and cancel itself (Operations).
* **Collaborators:** User (associated with), Website (associated with).

**Class: Website**

* **Responsibilities:** Knows Website **Attributes** (url, lastchecked). Can check for updates (checkforupdates() Operation) (Information Expert for checking itself). Knows its associated Subscriptions. Can create a Notification if an update is found (new Responsibility).
* **Collaborators:** Monitor (told to check), Subscription (associated with), Notification (creates).

**Class: Monitor**

* **Responsibilities:** Schedules and runs the check cycles (scheduleChecks(), runCheckCycle() Operations) (Controller for the checking process). Tells Website objects to check for updates. Handles the result of the check. Tells the NotificationService to send notifications if needed.
* **Collaborators:** Website (asks to check), NotificationService (tells to send notification).

**Class: Notification**

* **Responsibilities:** Knows Notification **Attributes** (notificationId, timestamp, content, status) (Information Expert about itself). Knows which Website and Subscription it is related to.
* **Collaborators:** NotificationService (is sent by).

**Class: NotificationService**

* **Responsibilities:** Sends a given Notification (Operation sendNotification()) (Information Expert for sending). Figures out the right way to send the notification based on user preferences.
* **Collaborators:** Notification (receives to send). User or Subscription (needs info to send).

**Did You Change Responsibilities?**

Yes, by using Controller and Information Expert:

* The Monitor Class clearly has the Controller job for the checking loop.
* The checkforupdates() **Operation** stays with Website because it's the Information Expert for its own status.
* Starting actions like registering or managing subscriptions might stay with User (acting as a simple Controller) or could move to a separate Controller Class for those specific tasks to improve the User Class's focus (Cohesion).
* The job of creating a Notification wasn't shown as an **Operation** in your original diagram, but it's needed. Based on these principles, Website (the expert about the change) or Monitor (the process controller) could be responsible for creating it (Creator principle).
* The NotificationService is the Information Expert for sending notifications.

By doing this, you are deliberately deciding which **Operations** belong to which Classes based on logical principles, which is a key part of good object-oriented design.