Worksheet

Extra Question 1

Question

Explain why the controllers are hard to test in the same way that you tested the `CacheProxy` (i.e. without relying on external services or views). Suggest how you might make a small modification to the controller class to make this easier.

Concise answer

Different from the test method of `CacheProxyTest`, which is used to test `CacheProxy`, all Controllers do not have a constructor. Based on the original code of Controllers, it is impossible to initialize variable parameters without relying on external services or views.

Detailed answer (Graphic collocation)

1. First, look at the `equalityTest()` part of `CacheProxyTest`. The test method is to verify whether `picture1` is the same as `picture2`, and they both come from the `Service` interface. (Figure 1)

2. Coming to the `Service` interface, my `getPicture()` found 12 ways to implement it. (Figure 2)

3. Back to `CacheProxyTest`, I make `getPicture()` return an empty picture, its role is to match the corresponding `subject` and `index`. (Figure 3)

4. Going back to `equalityTest()` again, we can find that it initializes the variable parameter. Coming to `CacheProxy`, we can find that the `CacheProxy` class implements Service. At the same time, it declares a `Service` interface type variable `baseService` which is initialized through two constructors. (Figure 4)

5. Next, look at the implementation method of `getPicture()`. As mentioned in the previous steps, the corresponding `subject` and `index` will return an empty picture. This empty picture exists in the cache, which is `picture1`, and the step of `picture2` will pass the `if` statement successfully without going through the `else` statement, so that the empty picture will be read from the cache and returned as `picture2` resulting in `picture1` and `picture2` being equal. (Figure 5)

6. In contrast to Controllers, I take `BigController` as an example where the `view` and `service` are private variables. We can't access them from the outside instead of using them in the class. However, `start()` does not pass parameters, and cannot modify the `view` and `service`, because the ones on the right are constants. (Figure 6)

Suggestion

A simple and feasible modification should be to add a constructor to the controller class so that it can modify the `view` and `service`.

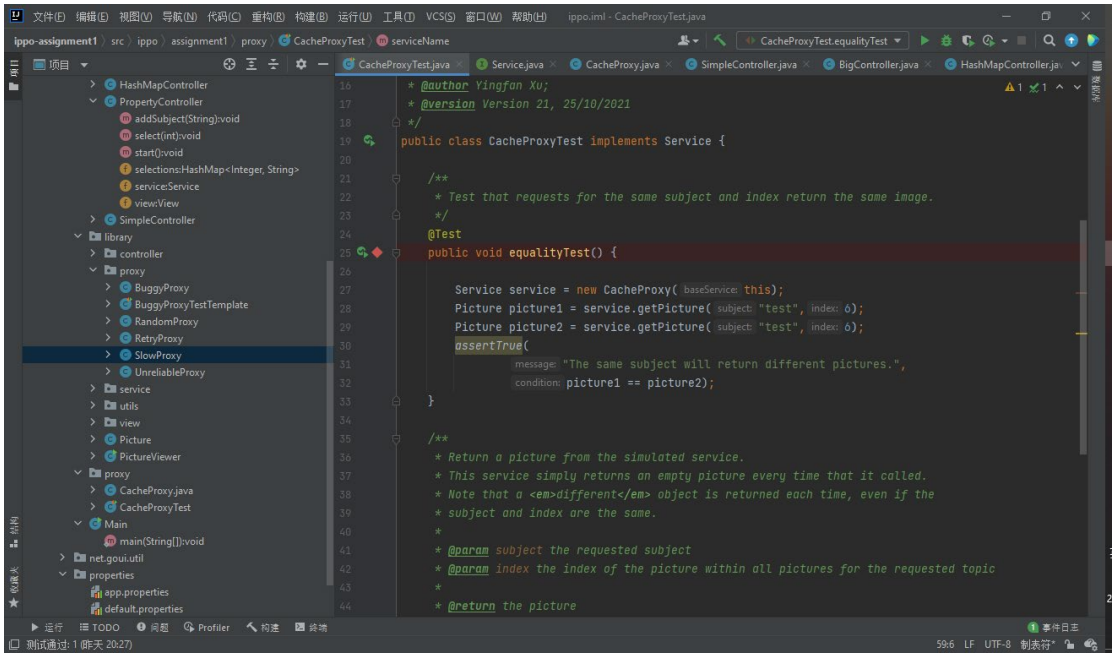


Figure 1. CacheProxyTest

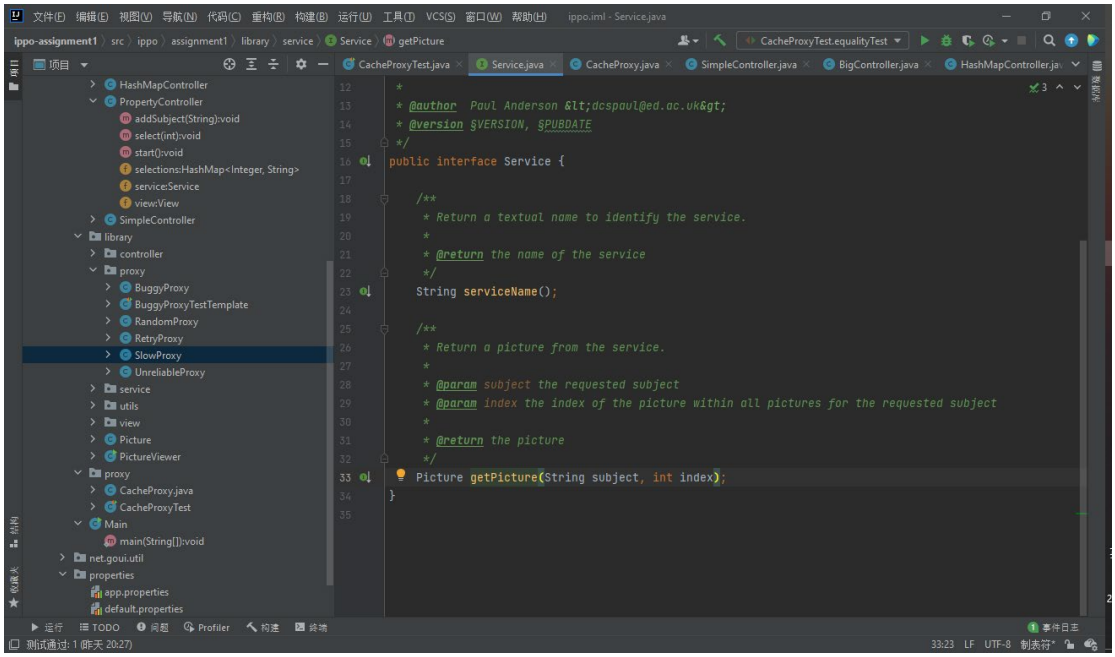


Figure 2. Service

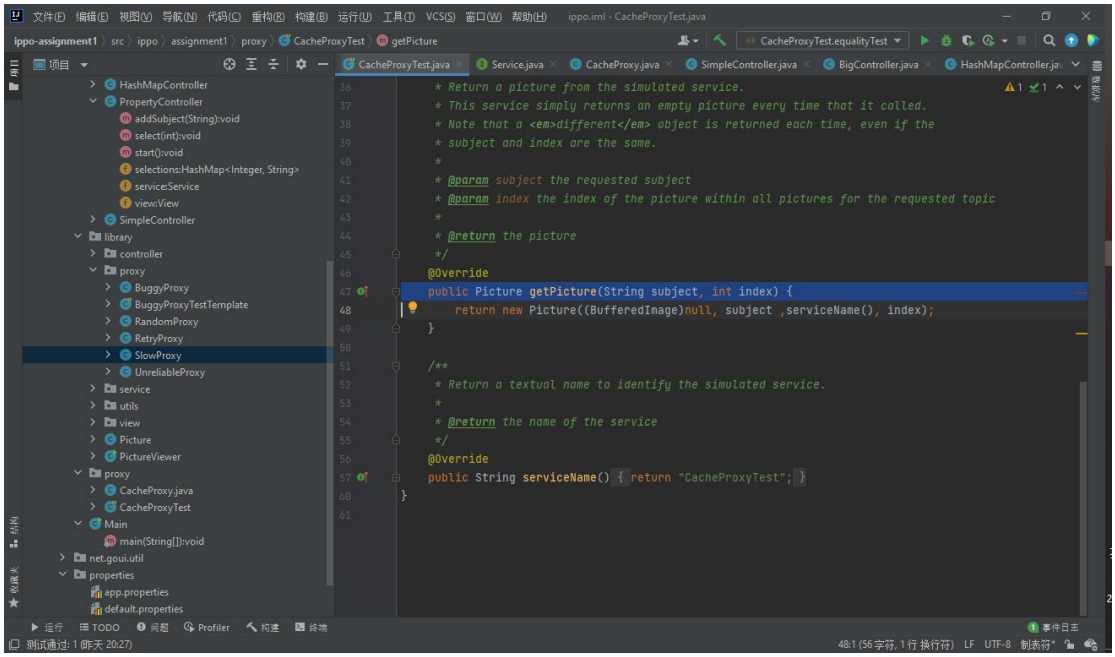


Figure 3. CacheProxyTest

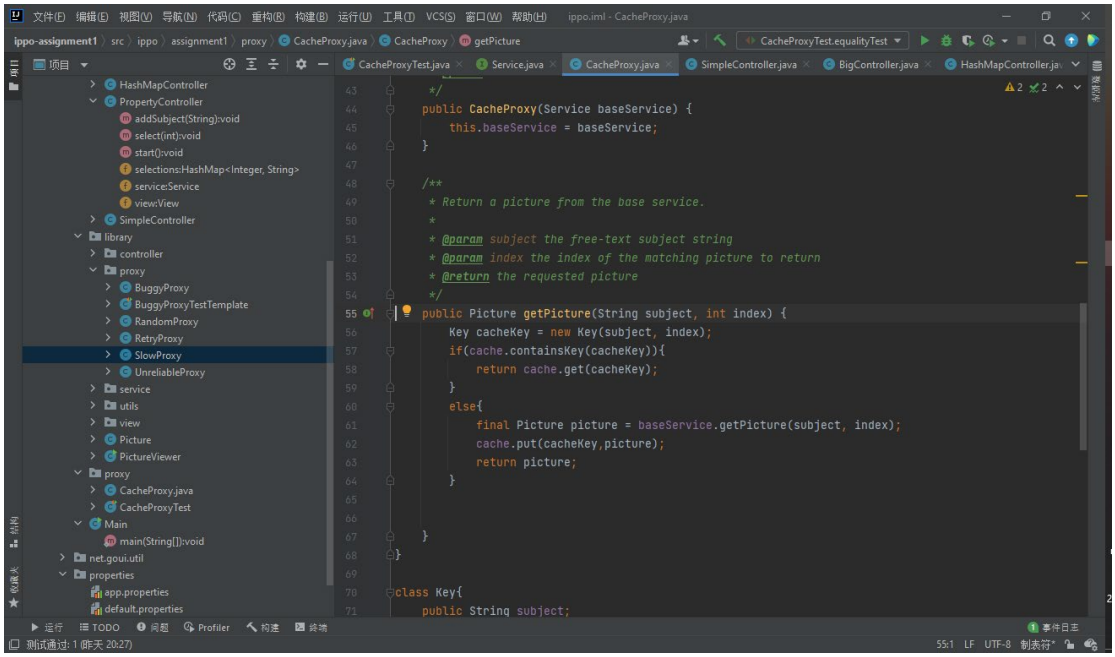


Figure 4. CacheProxy

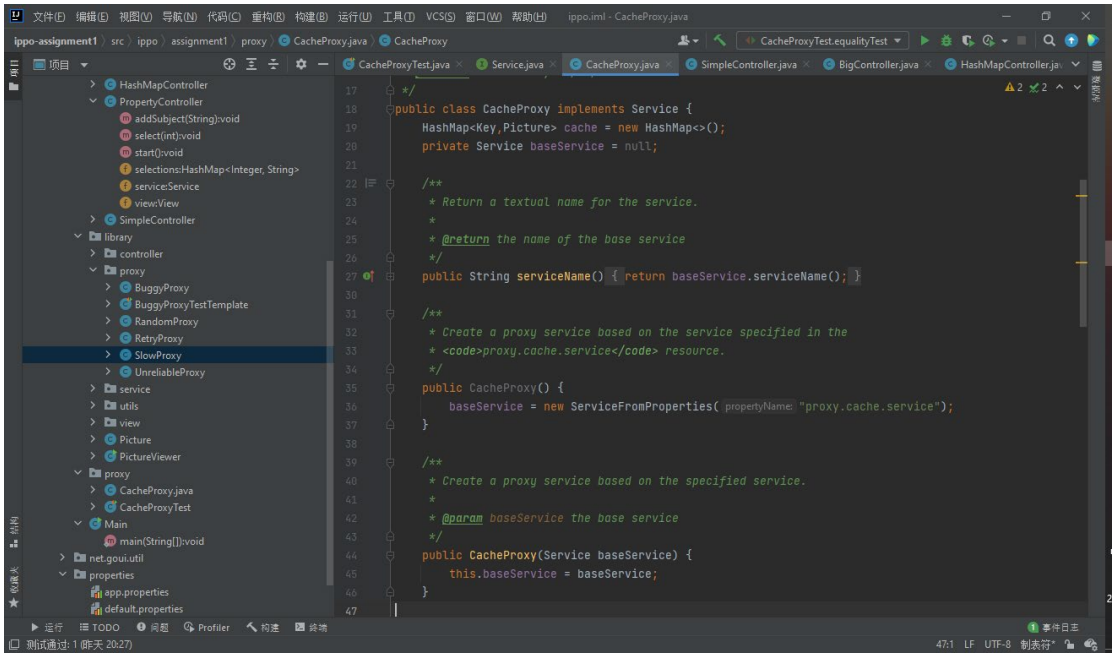


Figure 5. CacheProxy

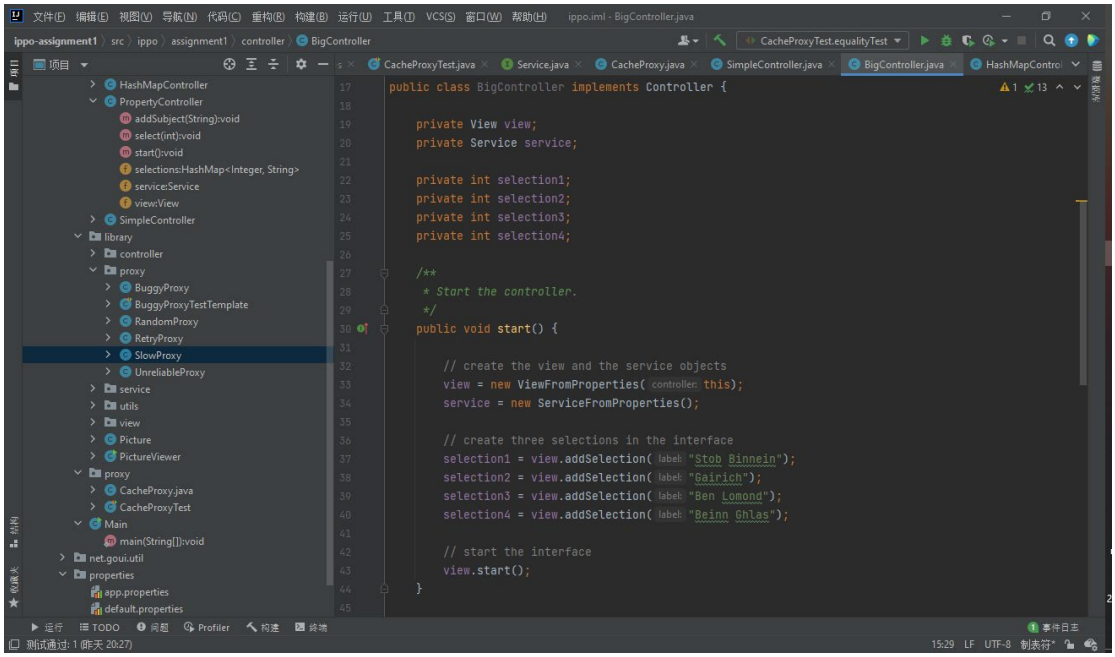


Figure 6. BigController

Extra Question 2

Question

Explain why it is difficult to "compose" the `RandomProxy` and `CacheProxy`. i.e. to use these class implementations to create a system which both fetches images from a random service and caches the results. Suggest how you might change the way in which the properties are used to make this composition possible

Concise answer

The logical sequence of the `if` and `else` statements used in `getPicture(String subject, int index)` cannot support `RandomProxy` to perform more than once (that is, more than or equal to twice). Once more than once, random results will no longer be generated, and subsequent results will always fetch the first random result from the cache.

Detailed answer (Graphic collocation)

1. First, according to a previous task, we know that `SlowProxy` and `CacheProxy` can be effectively composed through properties. (Figure 7)

2. Entering `SlowProxy`, we can find that its purpose is to delay, and it does not mind whether it will get pictures from the cache from the second run. (Figure 8)

3. Entering `RandomProxy`, we can find that its purpose is to get random results every time it runs. (Figure 9)

4. Finally, return to `CacheProxy`. According to the content in the `if` and `else` statements, I can find that the judgment of the cache is performed before the random result is generated, which contradicts the logical sequence implemented by `RandomProxy`. (Figure 10)

Suggestion

We need to rewrite a class, namely a new `getPicture(String subject, int index)`, so that it can generate a random result before judging whether the picture is in the cache.

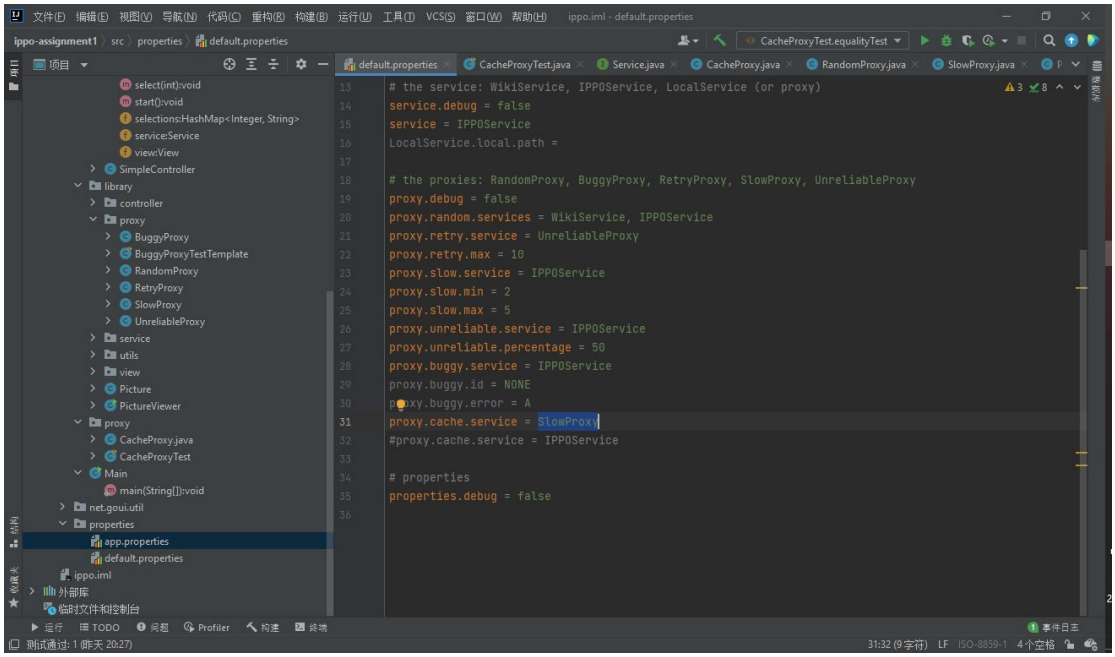


Figure 7. properties

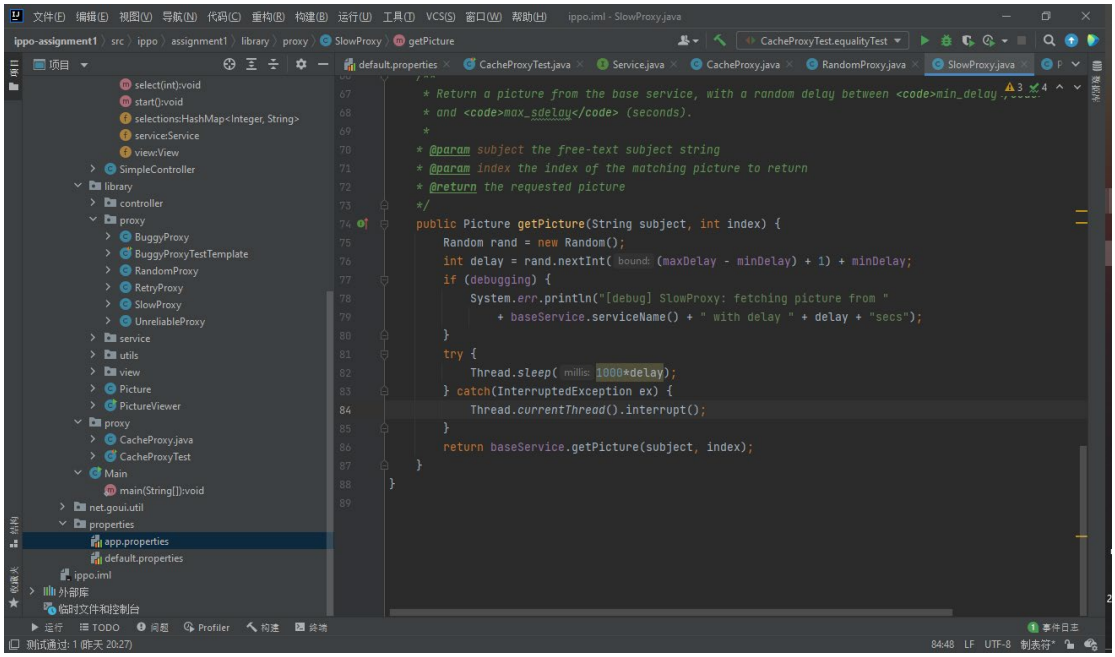
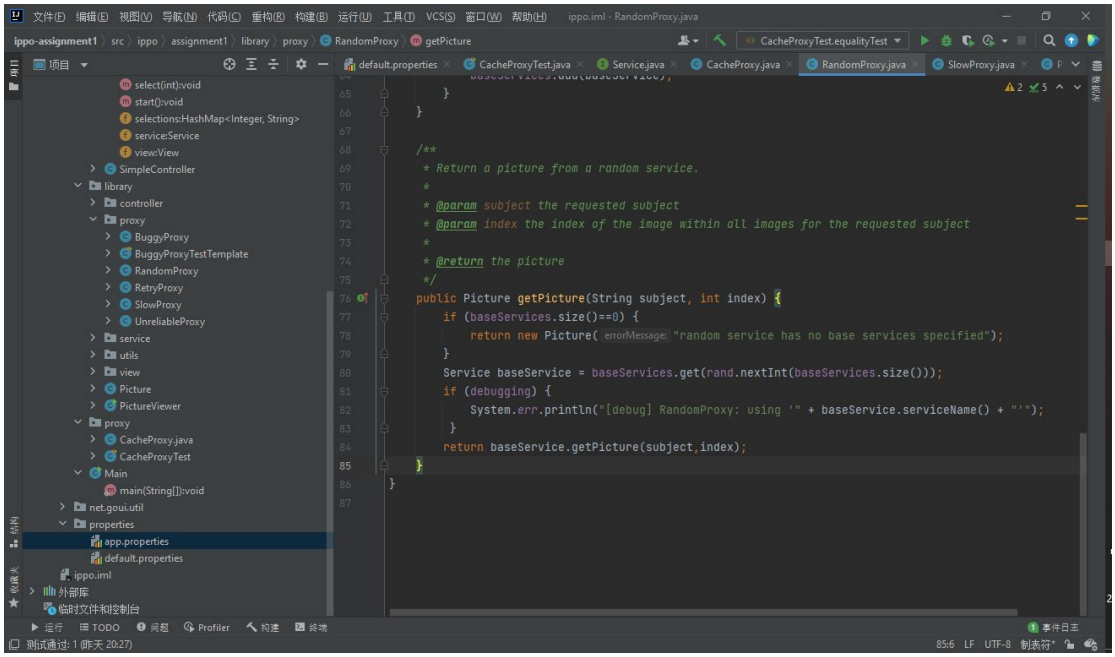


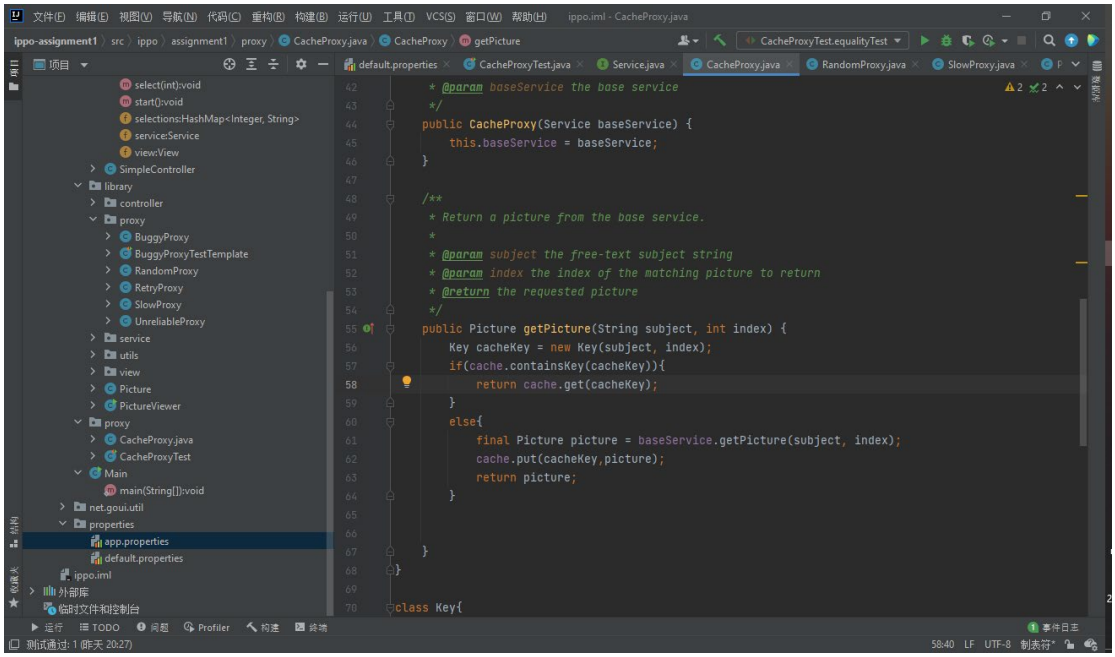
Figure 8. SlowProxy



```
    }
}

/**
 * Return a picture from a random service.
 *
 * @param subject the requested subject
 * @param index the index of the image within all images for the requested subject
 *
 * @return the picture
 */
public Picture getPicture(String subject, int index) {
    if (baseServices.size()==0) {
        return new Picture( errorMessage: "random service has no base services specified");
    }
    Service baseService = baseServices.get(rand.nextInt(baseServices.size()));
    if (debugging) {
        System.err.println("[debug] RandomProxy: using '" + baseService.serviceName() + "'");
    }
    return baseService.getPicture(subject,index);
}
}
```

Figure 9. RandomProxy



```
 * @param baseService the base service
 */
public CacheProxy(Service baseService) {
    this.baseService = baseService;
}

/**
 * Return a picture from the base service.
 *
 * @param subject the free-text subject string
 * @param index the index of the matching picture to return
 * @return the requested picture
 */
public Picture getPicture(String subject, int index) {
    Key cacheKey = new Key(subject, index);
    if(cache.containsKey(cacheKey)){
        return cache.get(cacheKey);
    }
    else{
        final Picture picture = baseService.getPicture(subject, index);
        cache.put(cacheKey,picture);
        return picture;
    }


}
}

class Key{
```

Figure 10. CacheProxy