

Using AI To Create Test Cases

- Example using ChatGPT
- On our example:

```
Status determineDiscount(  
    int bonusPoints,  
    boolean goldCustomer)
```

Enter the Java Code (copy-and- paste)

```
public class OnlineSales {  
  
    public static enum Status { FULLPRICE, DISCOUNT, ERROR };  
  
    /**  
     * Determine whether to give a discount for online sales.  
     * Gold customers get a discount above 80 bonus points.  
     * Other customers get a discount above 120 bonus points.  
     *  
     * @param bonusPoints How many bonus points the customer  
     * has accumulated  
     * @param goldCustomer Whether the customer is a Gold  
     * Customer  
     *  
     * @return  
     * DISCOUNT - give a discount<br>  
     * FULLPRICE - charge the full price<br>  
     * ERROR - invalid inputs  
     */  
    public static Status giveDiscount(long bonusPoints, boolean  
goldCustomer)  
    {  
        Status rv = FULLPRICE;  
        long threshold=120;  
  
        if (bonusPoints<=0)  
            rv = ERROR;  
  
        else {  
            if (goldCustomer)  
                threshold = 80;  
            if (bonusPoints>threshold)  
                rv=DISCOUNT;  
        }  
  
        return rv;  
    }  
}
```

Response (may differ per session)

The Java code you've provided defines a class called `OnlineSales` with an enum `Status` and a method `giveDiscount`. This method determines whether a customer should receive a discount based on their accumulated bonus points and whether they are a Gold customer.

Here is a quick breakdown of the key components:

- **Enum `Status`**: This enum defines three possible statuses: `FULLPRICE`, `DISCOUNT`, and `ERROR`.
- **Method `giveDiscount(long bonusPoints, boolean goldCustomer)`**:
 - This method checks if the customer qualifies for a discount based on their bonus points.
 - If the `bonusPoints` is greater than a set threshold, the customer gets a discount.
 - The threshold for Gold customers is 80 bonus points, while for other customers, it's 120 bonus points.
 - If the `bonusPoints` are less than or equal to 0, it returns an `ERROR` status.
 - Otherwise, it returns `DISCOUNT` if the points exceed the threshold, or `FULLPRICE` if they don't.

Response Continued

The Java code you've provided defines a class called `OnlineSales` with an enum `Status` and a method `giveDiscount`. This method determines whether a customer should receive a discount based on their accumulated bonus points and whether they are a Gold customer.

Here is a quick breakdown of the key components:

- **Enum `Status`**: This enum defines three possible statuses: `FULLPRICE`, `DISCOUNT`, and `ERROR`.
- **Method `giveDiscount(long bonusPoints, boolean goldCustomer)`**:
 - This method checks if the customer qualifies for a discount based on their bonus points.
 - If the `bonusPoints` is greater than a set threshold, the customer gets a discount.
 - The threshold for Gold customers is 80 bonus points, while for other customers, it's 120 bonus points.
 - If the `bonusPoints` are less than or equal to 0, it returns an `ERROR` status.
 - Otherwise, it returns `DISCOUNT` if the points exceed the threshold, or `FULLPRICE` if they don't.

Generate Unit Tests

write testng tests

Here is a set of **TestNG** test cases that can be used to verify the functionality of the `giveDiscount` method in the `OnlineSales` class. These test cases cover different scenarios for Gold customers and regular customers, including boundary conditions and invalid inputs.

The Tests (may differ per session)

java

Copy

```
import org.testng.Assert;
import org.testng.annotations.Test;
import example.OnlineSales;
import example.OnlineSales.Status;

public class OnlineSalesTest {

    @Test
    public void testGoldCustomerWithPointsAboveThreshold() {
        long bonusPoints = 100; // Gold customer threshold is 80 points
        boolean goldCustomer = true;

        Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);

        Assert.assertEquals(result, Status.DISCOUNT, "Gold customer with points above
    }

    @Test
    public void testGoldCustomerWithPointsBelowThreshold() {
        long bonusPoints = 70; // Gold customer threshold is 80 points
        boolean goldCustomer = true;

        Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);

        Assert.assertEquals(result, Status.FULLPRICE, "Gold customer with points below
    }
```

Continued

@Test

```
public void testNonGoldCustomerWithPointsAboveThreshold() {  
    long bonusPoints = 130; // Non-Gold customer threshold is 120 points  
    boolean goldCustomer = false;  
  
    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);  
  
    Assert.assertEquals(result, Status.DISCOUNT, "Non-Gold customer with points  
}
```

@Test

```
public void testNonGoldCustomerWithPointsBelowThreshold() {  
    long bonusPoints = 100; // Non-Gold customer threshold is 120 points  
    boolean goldCustomer = false;  
  
    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);  
  
    Assert.assertEquals(result, Status.FULLPRICE, "Non-Gold customer with point  
}
```

@Test

```
public void testInvalidBonusPointsZero() {  
    long bonusPoints = 0; // Invalid bonus points  
    boolean goldCustomer = true;  
  
    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);  
  
    Assert.assertEquals(result, Status.ERROR, "Zero bonus points should return  
}
```

Continued

```
@Test
public void testInvalidBonusPointsNegative() {
    long bonusPoints = -10; // Invalid bonus points
    boolean goldCustomer = false;

    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);

    Assert.assertEquals(result, Status.ERROR, "Negative bonus points should return error");
}

@Test
public void testGoldCustomerWithExactThreshold() {
    long bonusPoints = 80; // Gold customer threshold is exactly 80 points
    boolean goldCustomer = true;

    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);

    Assert.assertEquals(result, Status.FULLPRICE, "Gold customer with exactly 80 points should get full price");
}

@Test
public void testNonGoldCustomerWithExactThreshold() {
    long bonusPoints = 120; // Non-Gold customer threshold is exactly 120 points
    boolean goldCustomer = false;

    Status result = OnlineSales.giveDiscount(bonusPoints, goldCustomer);

    Assert.assertEquals(result, Status.FULLPRICE, "Non-Gold customer with exactly 120 points should get full price");
}
```


Explanation

Explanation of Test Cases:

1. **testGoldCustomerWithPointsAboveThreshold:** Verifies that a Gold customer with more than 80 points gets a discount.
2. **testGoldCustomerWithPointsBelowThreshold:** Verifies that a Gold customer with fewer than 80 points does not get a discount.
3. **testNonGoldCustomerWithPointsAboveThreshold:** Verifies that a non-Gold customer with more than 120 points gets a discount.
4. **testNonGoldCustomerWithPointsBelowThreshold:** Verifies that a non-Gold customer with fewer than 120 points does not get a discount.
5. **testInvalidBonusPointsZero:** Verifies that invalid bonus points (0) return an error.
6. **testInvalidBonusPointsNegative:** Verifies that invalid bonus points (negative) return an error.
7. **testGoldCustomerWithExactThreshold:** Verifies that a Gold customer with exactly 80 points does not get a discount.
8. **testNonGoldCustomerWithExactThreshold:** Verifies that a non-Gold customer with exactly 120 points does not get a discount.

You can run these tests using TestNG to ensure the `giveDiscount()` method behaves as expected in different scenarios.

Ask ChatGPT

explain the test technique used

The test technique used in the **TestNG test cases** for the `determineDiscount` method is **unit testing**. The key objective is to test individual units of the code (in this case, the `determineDiscount` method) to ensure that they function correctly under different conditions. The following techniques are used:

Answer (Session 1) - details omitted

Summary of Techniques:

1. **Boundary Testing:** Checking the behavior at the boundary values (e.g., 80 and 120 bonus points).
2. **Equivalence Partitioning:** Grouping similar input values into partitions, assuming if one value in the partition works, others should work too.
3. **Positive Testing:** Verifying the method works as expected for valid, correct inputs.
4. **Negative Testing:** Ensuring the method handles invalid inputs (e.g., negative bonus points).
5. **Code Coverage:** Ensuring that the method covers all branches and edge cases in the code.
6. **Assertions:** Verifying the results with the expected output.

By combining these techniques, the tests ensure that the `determineDiscount` method behaves correctly for both valid and invalid inputs, under normal and edge cases, and in all logical branches of the code.

Answer (Session 2)

The test technique used in the provided TestNG tests is **Boundary Value Analysis (BVA)**, **Equivalence Partitioning (EP)**, and **Error Guessing**. These techniques are common in software testing for identifying potential edge cases, ensuring correct behavior for valid inputs, and testing invalid conditions.

1. Boundary Value Analysis (BVA):

- **Definition:** BVA focuses on testing the boundary values (limits) of input ranges. The idea is that defects are often found at the boundaries of input ranges rather than within the middle.
- **Examples from the tests:**
 - **Test with exact boundary values:**
 - `testGoldCustomerWithExactThreshold` : Tests exactly 80 bonus points for a Gold customer, which is the threshold.
 - `testNonGoldCustomerWithExactThreshold` : Tests exactly 120 bonus points for a non-Gold customer, which is the threshold.
 - **Test with values just above or below the boundary:**
 - `testGoldCustomerWithPointsAboveThreshold` : Tests a Gold customer with 100 points (above the threshold of 80).
 - `testNonGoldCustomerWithPointsAboveThreshold` : Tests a non-Gold customer with 130 points (above the threshold of 120).

Answer continued (Session 2)

2. Equivalence Partitioning (EP):

- **Definition:** EP is used to divide input data into valid and invalid partitions. The assumption is that all values within a partition will behave similarly.
- **Examples from the tests:**
 - **Valid equivalence classes:**
 - Gold customer with points above 80 (`testGoldCustomerWithPointsAboveThreshold`).
 - Non-Gold customer with points above 120 (`testNonGoldCustomerWithPointsAboveThreshold`).
 - **Invalid equivalence classes:**
 - Bonus points equal to 0 or negative (`testInvalidBonusPointsZero` and `testInvalidBonusPointsNegative`).
 - **Valid equivalence classes** for exact threshold checks:
 - Exact points that are not eligible for a discount, such as 80 for Gold customers and 120 for non-Gold customers.

Answer continued (Session 2)

3. Error Guessing:

- **Definition:** Error Guessing involves using the tester's experience and intuition to guess problematic areas in the code that might break or cause errors.
- **Examples from the tests:**
 - **Invalid input handling:** The tests `testInvalidBonusPointsZero` and `testInvalidBonusPointsNegative` are based on error guessing, as invalid input values (such as 0 or negative points) are likely to result in errors or incorrect behavior.
 - **Edge case for exact thresholds:** Test cases like `testGoldCustomerWithExactThreshold` and `testNonGoldCustomerWithExactThreshold` attempt to identify edge cases where the logic might fail, such as returning a discount when the bonus points are precisely at the threshold.

Summary:

- **Boundary Value Analysis** is used to test the boundary limits (e.g., 80 points for Gold customers and 120 points for non-Gold customers).
- **Equivalence Partitioning** is used to test valid and invalid input ranges (e.g., valid points for a discount and invalid negative or zero points).
- **Error Guessing** is used to predict and test potential failure points in the code, such as handling of invalid inputs.

By applying these techniques, the test suite covers a wide range of scenarios to ensure that the `giveDiscount()` method functions correctly in both normal and edge cases.

IMPORTANT NOTE

- You get different tests generated each time!!

Question

- Is this adequate testing?
- Would you trust ChatGPT to write the tests?
- How would you check their correctness and completeness?

Test Review

EP coverage?

BVA coverage?

Combinations coverage?

OWNERSHIP

When you use an AI assistant (ChatGPT, CoPilot, etc.)
You must take ownership of the tests

For testing: review the tests
And ensure coverage

Make sure the code executes!

Modify if necessary