Diagnostic tests

Contents

Features to assess in the diagnostic test

1. Test coverage (i.e. robustness, not only happy path).
2. Readable code + code legacy (i.e. refactoring, style).
3. Translate from natural language test to specifications.
4. Definition of a class from a text.

Tests cases exercises

1. We want to create a code that calculates the factorial of a number in the range of 0 and 12, both included.

```
public int factorial (int number) {
  if (number<0 || number>=12){
      System.out.println("Error: number out of range");
      return -1;
  }else  if (number==0){
      return 1;
  }else{
     return number * factorial(number-1);
  }
}
```

Indicate three test cases that we must to create in order to check the correctness of the program:

- **Handle negative numbers.**
- **Handle numbers higher than 12.**
- **Corner cases: 0 and 12.**
- **Correct cases: 1, 2, 3, 12, 33, etc.**

2. We want to create a code that calculates the Fibonacci of a positive number, being 0 included.

```
public int fibonacci(int n){

  if (n>1){
   return fibonacci(n-1) + fibonacci(n-2);
  }else if (n==1) {
    return 1;
  }else if (n==0){
    return 0;
  }else{ //error
    System.out.println("You must enter a positive number!!");
    return -1;
  }
}
```

Indicate what test cases you will use in order to check the correctness of the program:

- **Handle negative numbers.**
- **Corner cases: 0 and 1.**
- **Correct cases: 2, 3, 12, 33, etc.**

Find semantic style errors

1. The following code has some semantic style errors.

```
public int specialFunction(String word, int times) {
  int length;
  length = (int) (word.length() * times);

  if(word.length() * times % 2 == 0){
   System.out.println("New length: "+length);
   return true;
  }else if(word.length() * times % 2 != 0){
   System.out.println("New length: "+length);
   return false;
  }
}
```

Which are they?

Variable declaration and assignation in different lines.

Unneeded casting.

Repeated expression in the conditional

Opposite expression in the else-if conditional.

Repeated `println` in different branches.

Return with `true/false` when the answer is the condition itself

Refactor the code:

```
public int specialFunction(String word, int times) {
  int length = (word.length() * times);

  System.out.println("New length: "+length);

  return (length % 2 == 0);
}
```

Refactoring exercises

1. Refactor the following code.

```
public Position walkRight () {
  Player player = getPlayer();
  player.move("R");
  return player.getPosition();
}

public Position walkLeft () {
  Player player = getPlayer();
  player.move("L");
  return player.getPosition();
}
```

```
public Position walk (String direction) {
  Player player = getPlayer();
  player.move(direction);
  return player.getPosition();
}
```

2. Refactor the following code.

```
public sumEvenValues(int[]values){
 int i=0;
 int sum = i;

 while(i<values.length){
  if(values[i] % 2 == 0) {
   sum += values[i];
   i++;
  }else {
   i++;
  }
 }
}
```

Option 1

```
public sumEvenValues(int[]values){
 int i=0;
 int sum = 0;

 while(i<values.length){
  if(values[i] % 2 == 0) {
   sum+=values[i];
  }
  i++;
 }
}
```

Option 2

```
public sumEvenValues(int[]values){
 int i=0;
 int sum = 0;

 for(int i=0; i<values.length; i++){
  if(values[i] % 2 == 0) {
   sum+=values[i];
  }
 }
}
```

3. Refactor the following code.

```
public getPay(int age, int normalHours, int overtimeHours){

 if (age <= 20) {
  int payRate = 15;
  int overtimeRate = payRate * 2;
  int pay =
(normalHours*payRate)+(overtimeHours*overtimeRate);
 return pay;

 }else{
  int payRate = 35;
  int overtimeRate = payRate * 2;
  int pay = (normalHours * payRate)+(overtimeHours *
overtimeRate);
  return pay;
 }
}
```

```
public getPay(int age, int normalHours, int overtimeHours){

 int payRate = 35;
 int overtimeRate = 0;

 if (age <= 20) {
  payRate = 15;
 }

 overtimeRate = payRate * 2;

 return(normalHours*payRate)+(overtimeHours*overtimeRate);
}
```

4. Refactor the following code.

```
public boolean isExpensive(int threshold) {
  int expensiveThreshold;
  expensiveThreshold = threshold;

  if(this.getValue()>expensiveThreshold){
   return true;
  }else(this.getValue()<=expensiveThreshold){
   return false;
  }
}
```

```
public boolean isExpensive() {
  return this.getValue()>threshold;
}
```

5. Given the following code.

```
public class Product{
  private int quantity;
  private double itemPrice;

  public Product(int quantity, double itemPrice){
    if(quantity>=0){
      this.quantity = quantity;
    }else{
      System.out.println("Quantity error!");
    }

    if(itemPrice>=0){
      this.itemPrice = itemPrice;
    }else{
      System.out.println("Item's price error!");
    }
  }

  public double calculateTotal(){
    double basePrice;
    basePrice = quantity * itemPrice;

     if(basePrice>60){
      int totalPrice = basePrice * 0.25;
      return totalPrice;
     }else if(basePrice>30){
      int totalPrice = basePrice * 0.15;
      return totalPrice;
     }else{
      return basePrice;
     }
  }
```

```
public class Product{
  private int quantity;
  private double itemPrice;

  public Product(int quantity, double itemPrice){
    setQuantity(quantity);
    setItemPrice(itemPrice);
  }

 private void setQuantity(int quantity){
    if(quantity>=0){
      this.quantity = quantity;
    }else{
      System.out.println("Quantity error!");
    }
 }

private void setItemPrice(int itemPrice){
    if(itemPrice >=0){
      this.itemPrice = itemPrice;
    }else{
      System.out.println("Item's price error!");
    }
}

private double getBasePrice(){
     return quantity * itemPrice;
}


public double calculateTotal(){
    double basePrice = basePrice();

    if(basePrice>60){
      return basePrice * 0.25;
    }else if(basePrice>30){
      return basePrice * 0.15;
    }else{
      return basePrice;
    }
}
```