# Testing Example
***Category-Partition Method***

This example will use the spirit of ECT and C-P to develop several unit tests.

## Problem Statement

Consider this problem from CS 1302.

You will write a class named *Employee* that keeps track of the hours worked on each day of the week. The class will have the following members:

a. hours – a double array with 7 elements. This holds the hours worked on each day of the week. Monday (index=0) is the first day of the week.
b. name – a string containing the name of the employee
c. payRate – a double which represents the pay rate ($/hr)
d. Employee(name:String, payRate:double) – constructor that accepts a name for the employee and their pay rate ($/hr). The name must have a length greater than zero and the pay rate must be greater than zero; otherwise, a *RuntimeException* is thrown.
e. getHours(i:int) – returns the number of hours worked on day i.
f. getName – returns the name
g. getNumDaysWorked – returns the number of days worked.
h. getPay – returns the total pay for the week computed in the following way:
    - Weekday hours (Mon-Fri) are paid at the pay rate. Any hours over 40 during weekdays are paid at time-and-a-half.
    - Weekend hours are paid at double-time, no matter whether the workday hours exceeds 40.
    - Working 7 consecutive day earns a bonus of $50.00, no matter how many total hours worked.
i. getPayRate – returns the pay rate
j. getTotalHours – returns the total number of hours worked for the week
k. getWeekdayHours – returns the total number of hours worked during weekdays (Mon-Fri)
l. getWeekendHours – returns the total number of hours worked during the weekend (Sat-Sun)
m. mergeEmployee – accepts an *Employee* object and merges the hours. You can assume the input employee has the same name and pay rate. For example if *e1* had hours: [8,8,8,2,0,0,0] and *e2* had hours: [0,0,0,4,10,0,0]. Then, when *e1.mergeEmployee(e2)* is excecuted, the *e1* will have hours: [8,8,8,6,10,0,0].
n. newWeek – starts the week over by setting 0 hours for each day
o. setHours(i:int,num:double) – sets the number of hours worked on day i.
p. toString – returns a string that is formatted like this…

## Unit Test for Constructor

`Employee(name:String, payRate:double)`

1. Input, Categories, and Choices

    Inputs
    - Name – length
      - =0 ERROR
      - >0
    - Pay rate – value
      - <0 ERROR
      - =0 ERROR
      - >0

2. Test Specification

| Num | Name | Pay rate |
|---|---|---|
| 1 | =0 | >0 |
| 2 | >0 | <0 |
| 3 | >0 | =0 |
| 4 | >0 | >0 |

3. Test Cases

| Num | Name | Pay rate | Expected Result |
|---|---|---|---|
| 1 | “” | 50.0 | Exception |
| 2 | “Markus” | -10.0 | Exception |
| 3 | “Markus” | 0.0 | Exception |
| 4 | “Markus” | 50.0 | Getters produce correct result |

4. Test Cases

```
@DisplayName("Constructor: Empty string")
@Test
void testConstructorEmptyString() {
    Assertions.assertThrows(RuntimeException.class, () -> {new Employee("",50.0);});
}
@DisplayName("Constructor: Payrate <0")
@Test
void testConstructorPayrateLess0() {
    Assertions.assertThrows(RuntimeException.class, () -> {new Employee("Markus",-10.0);});
}

@DisplayName("Constructor: Payrate =0")
@Test
void testConstructorPayrateEqual0() {
    Assertions.assertThrows(RuntimeException.class, () -> {new Employee("Markus",0.0);});
}

@DisplayName("Constructor: saves instance vars")
```

```
@Test
void testConstructorNormal() {
    Employee e = new Employee("Markus", 50.0);
    assertEquals("Markus",e.getName());
    assertEquals(50.0,e.getPayRate());
}
```

## Unit Test for getPay()

`getPay():double`

1. Reread the specification for this method, it is fairly detailed. Exactly what is the input for this method – after all the method doesn't have any formal parameters? But it does use the instance variables: *hours* array and the *payRate.* However, *payRate* is not really a factor since we already tested that it is >0. Thus, the equivalence class/choice for pay-rate is simply: >0 and a single pay rate will suffice for testing.

2. For the *hours* array, initially, I decided the categories should be (after several rounds writing things down, scratching out, *etc.*):

| | **Category** | | |
|---|---|---|---|
| | **Over-time Pay** | **Double-time Pay** | **7-Day Bonus Pay** |
| **Choices** | No, Yes | No, Yes | No, Yes |

This lead to the following combinations:

| **Num** | **OT** | **DT** | **7-Day** | **Comment** |
|---|---|---|---|---|
| 1 | N | N | N | |
| | N | N | Y | Not Possible |
| 2 | N | Y | N | |
| 3 | N | Y | Y | |
| 4 | Y | N | N | |
| | Y | N | Y | Not Possible |
| 5 | Y | Y | N | |
| 6 | Y | Y | Y | |

I decided that this was one category for *hours* with 6 possible choices. I'll call this category: *Pay Source* (PS)

3. Next, I decided that I needed 3 more categories for the *hours* array:

   - WDH – Week-day hours worked: {<40, =40, >40}
   - WDW – Week-days worked: {0,4,5}
   - WEW – Week-end days worked: {0,1,2}

   Thus, with the first category:

   - PS – Pay Source {NNN, NYN, NYY, YNN, YYN, YYY}

   there are 3*3*3*6 = 162 combinations. Still, many of them aren't possible.

4. Next, I constructed the TSL File. Slowly, I added constraints using knowledge of the problem to reduce the number of test frames.

```
#Week Days Worked
Value:
  0.  [property WDW=0]
  4.  [property WDW=4]
  5.  [property WDW=5]
#Week Day Hours
Value:
  <40.
  =40.  [if !WDW=0]
  >40.  [if !WDW=0][property WDH>40]
#Weekend Days Worked
Value:
  0.  [property WEDW=0]
  1.  [property WEDW=1]
  2.  [property WEDW=2]
#PaySource
Type:
  NNN.  [if !WDH>40 && WEDW=0]
  NYN.  [if !WDH>40 && ((WDW=4 && (WEDW=1 || WEDW=2)) || (WDW=5 && WEDW=1))]
  NYY.  [if !WDH>40 && WDW=5 && WEDW=2]
  YNN.  [if (WDH>40 && !WDW=0) && WEDW=0]
  YYN.  [if (WDH>40 && ((WDW=4 && (WEDW=1 || WEDW=2)) || (WDW=5 && WEDW=1)))]
  YYY.  [if WDH>40 && WDW=5 && WEDW=2]
```

   When this file is run with TSL, there are 21 test frames.

5. Let's consider a few of the constraints in detail.

   a. No overtime, no double time, no 7-day bonus. Employee works <= 40 hours during the week and no days on the weekend.

```
  NNN.  [if !WDH>40 && WEDW=0]
```

   b. No overtime, did get double-time, but not 7-day bonus. Employee works <= 40 hours during the week over 4 weekdays and works 1 or 2 weekend days. Or, Employee works <=40 hours over 5 weekdays and works 1 weekend day.

```
  NYN.  [if !WDH>40 && ((WDW=4 && (WEDW=1 || WEDW=2)) || (WDW=5 && WEDW=1))]
```

c. No overtime, but did get both double-time and 7-day bonus. Employee works <= 40 hours during the week over 5 weekdays, and worked 2 weekend days.

```
NYY.  [if !WDH>40 && WDW=5 && WEDW=2]
```

d. Got overtime, but not double time nor 7-day bonus. Employee works >40 hours during the week and did not work on weekend (and worked more than zero days).

```
YNN.  [if (WDH>40 && !WDW=0) && WEDW=0]
```

e. Got overtime and double time, but not 7-day bonus. Employee works >40 hours during the week over 4 days and either 1 or 2 weekend days. Or, Employee works >40 hours during the week over 5 days and 1 day on weekend.

```
YYN.  [if (WDH>40 && ((WDW=4 && (WEDW=1 || WEDW=2)) || (WDW=5 && WEDW=1)))]
```

f. Got overtime, double time, and 7-day bonus. Employee works >40 hours during the week over 5 days and either both weekend days.

```
YYY.  [if WDH>40 && WDW=5 && WEDW=2]
```

## 6. Test Frames

Test Case 1 (Key = 1.1.1.1.)
  Value : 0
  Value : <40
  Value : 0
  Type : NNN

Test Case 2 (Key = 1.1.2.0.)
  Value : 0
  Value : <40
  Value : 1
  Type : <n/a>

Test Case 3 (Key = 1.1.3.0.)
  Value : 0
  Value : <40
  Value : 2
  Type : <n/a>

Test Case 4 (Key = 2.1.1.1.)
  Value : 4
  Value : <40
  Value : 0
  Type : NNN

Test Case 5 (Key = 2.1.2.2.)
  Value : 4
  Value : <40
  Value : 1
  Type : NYN

Test Case 6 (Key = 2.1.3.2.)
  Value : 4
  Value : <40
  Value : 2
  Type : NYN

Test Case 7 (Key = 2.2.1.1.)
  Value : 4
  Value : =40
  Value : 0
  Type : NNN

Test Case 8 (Key = 2.2.2.2.)
  Value : 4
  Value : =40
  Value : 1
  Type : NYN

Test Case 9 (Key = 2.2.3.2.)
  Value : 4
  Value : =40
  Value : 2
  Type : NYN

Test Case 10 (Key = 2.3.1.4.)
  Value : 4
  Value : >40
  Value : 0
  Type : YNN

Test Case 11 (Key = 2.3.2.5.)
  Value : 4
  Value : >40
  Value : 1
  Type : YYN

Test Case 12 (Key = 2.3.3.5.)
  Value : 4
  Value : >40
  Value : 2
  Type : YYN

Test Case 13 (Key = 3.1.1.1.)
  Value : 5
  Value : <40
  Value : 0
  Type : NNN

Test Case 14 (Key = 3.1.2.2.)
  Value : 5
  Value : <40
  Value : 1
  Type : NYN

Test Case 15 (Key = 3.1.3.3.)
  Value : 5
  Value : <40
  Value : 2
  Type : NYY

Test Case 16 (Key = 3.2.1.1.)
  Value : 5
  Value : =40
  Value : 0
  Type : NNN

Test Case 17 (Key = 3.2.2.2.)
  Value : 5
  Value : =40
  Value : 1
  Type : NYN

Test Case 18 (Key = 3.2.3.3.)
  Value : 5
  Value : =40
  Value : 2
  Type : NYY

Test Case 19 (Key = 3.3.1.4.)
  Value : 5
  Value : >40
  Value : 0
  Type : YNN

Test Case 20 (Key = 3.3.2.5.)
  Value : 5
  Value : >40
  Value : 1
  Type : YYN

Test Case 21 (Key = 3.3.3.6.)
  Value : 5
  Value : >40
  Value : 2
  Type : YYY

7. Summary of Test frames.

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 1 | NNN | <40 | 0 | 0 |
| 2 | | <40 | 4 | 0 |
| 3 | | <40 | 5 | 0 |
| 4 | | =40 | 4 | 0 |
| 5 | | =40 | 5 | 0 |

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 6 | NYN | <40 | 0 | 1 |
| 7 | | <40 | 0 | 2 |
| 8 | | <40 | 4 | 1 |
| 9 | | <40 | 4 | 2 |
| 10 | | <40 | 5 | 1 |
| 11 | | =40 | 4 | 1 |
| 12 | | =40 | 4 | 2 |
| 13 | | =40 | 5 | 1 |

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 14 | NYY | <40 | 5 | 2 |
| 15 | | =40 | 5 | 2 |

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 16 | YNN | >40 | 4 | 0 |
| 17 | | >40 | 5 | 0 |

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 18 | YYN | >40 | 4 | 1 |
| 19 | | >40 | 4 | 2 |
| 20 | | >40 | 5 | 1 |

| Num | PS | WDH | WDW | WEW |
|---|---|---|---|---|
| 21 | YYY | >40 | 5 | 2 |

8. Next, you would have to construct the array for each of these 21 cases and write unit-tests which you will do in a homework assignment.