Object Oriented Software Engineering
Tutorial on Software Testing

Fani Deligianni,
Lecturer
School of Computing Science,
University of Glasgow

Multiple Choice Question 1

Which of the following would be a valid measure of test progress

A. Number of undetected defects
B. Number of test cases not yet executed
C. Total number of defects in the product
D. Effort required to fix all defects

Multiple Choice Question 1

Which of the following would be a valid measure of test progress

A. Number of undetected defects
B. Number of test cases not yet executed 
C. Total number of defects in the product
D. Effort required to fix all defects

Multiple Choice Question 2

Statement Coverage will not check for the following:

A. Missing Statements
B. Unused Branches
C. Dead Code
D. Unused Statement

Multiple Choice Question 2

Statement Coverage will not check for the following:

A. Missing Statements ✓
B. Unused Branches
C. Dead Code
D. Unused Statement

Exercise 1: Unit Testing

```
public class ProcessA {
    private int i = 0;
    @Override
    public boolean equals(Object obj) {
        return (this==obj);
    }

}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- What is going to be the output of ‘assert Same’ and ‘assertEqual’ in the above scenario and why?

Solution 1: Unit Testing

```
public class ProcessA {
    private int i = 0;
    @Override
    public boolean equals(Object obj) {
        return (this==obj);
    }

}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- ‘assertSame’ would fail because by default it compares the references of the two objects.

Solution 1: Unit Testing

```
public class ProcessA {
    private int i = 0;
    @Override
    public boolean equals(Object obj) {
        return (this==obj);
    }

}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- ‘assertSame’ would fail because by default it compares the references of the two objects.
- ‘assertEqual’ would fail because here it compares the reference of the two class objects.

Exercise 2: Unit Testing

```
public class ProcessA {
    Random rd = new Random();
    public int i = rd.nextInt();
    @Override
    public boolean equals(Object obj) {
        return (this.i==obj.i);
    }
}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- What is going to be the output of ‘assert Same’ and ‘assertEqual’ in the above scenario and why?

Solution 2: Unit Testing

```
public class ProcessA {
    Random rd = new Random();
    public int i = rd.nextInt();
    @Override
    public boolean equals(Object obj) {
        return (this.i==obj.i);
    }
}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- ‘assertSame’ would fail because by default it compare the references of the two objects.

Solution 2: Unit Testing

```
public class ProcessA {
    Random rd = new Random();
    public int i = rd.nextInt();
    @Override
    public boolean equals(Object obj) {
        return (this.i==obj.i);
    }
}
```

```
ProcessA myProcess1 = new ProcessA();
ProcessA myProcess2 = new ProcessA();
assertSame(myProcess1,myProcess2);
assertEqual(myProcess1,myProcess2);
```

- ‘assertSame’ would fail because by default it compare the references of the two objects.
- ‘assertEqual’ most probably will fail, since it is unlikely the two objects to have been initialised with the same value.

Exercise 3: Control Flow Graphs

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8              z = z * x;
9              pow = pow - 1;
10     }
11     if (y < 0){
12             z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

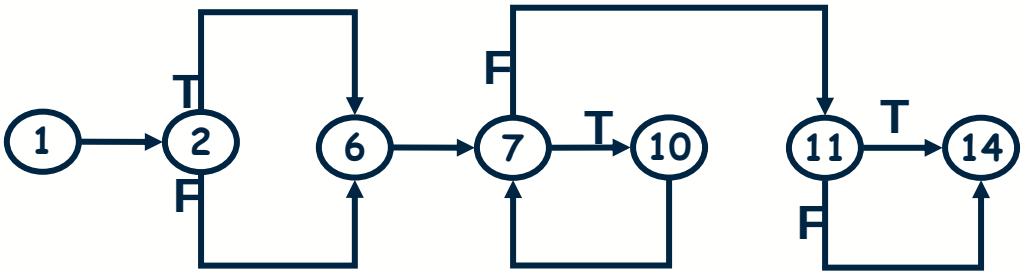
a. Create the control flow graph of the above code

b. Examine statement, branch and decision coverage for cases:

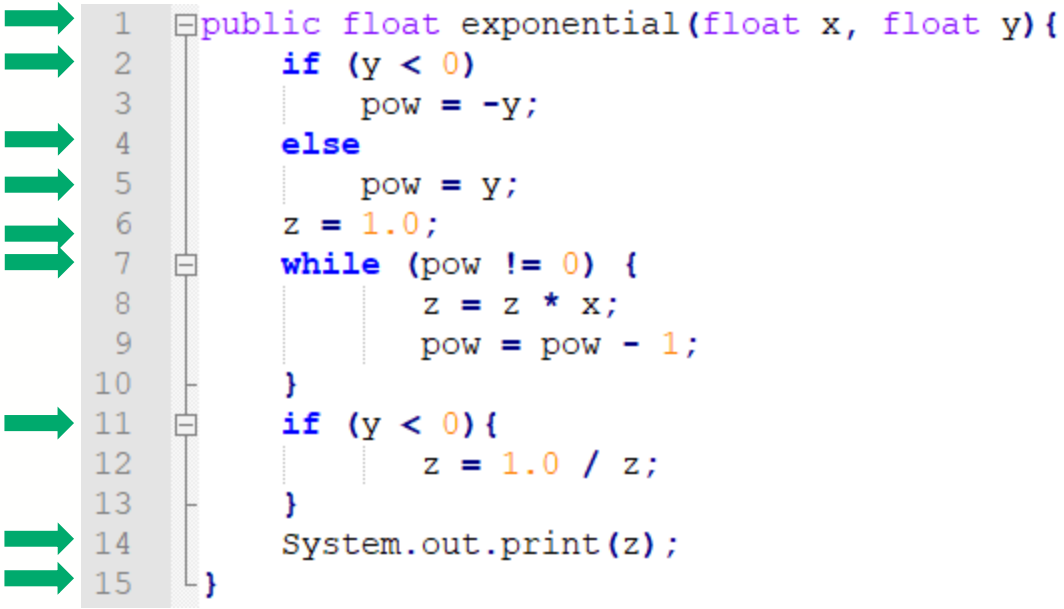
- (x,y)– [(0,0)]
- (x,y)-[(0,0),(0,1)]

Solution 3a): Control Flow Graphs

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8              z = z * x;
9              pow = pow - 1;
10     }
11     if (y < 0){
12             z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```



Solution 3b: Statement Coverage



```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

For test case (x,y)-[(0,0)]

Number of statements covered: 9

Statement coverage = 9/15

Total number of statements: 15

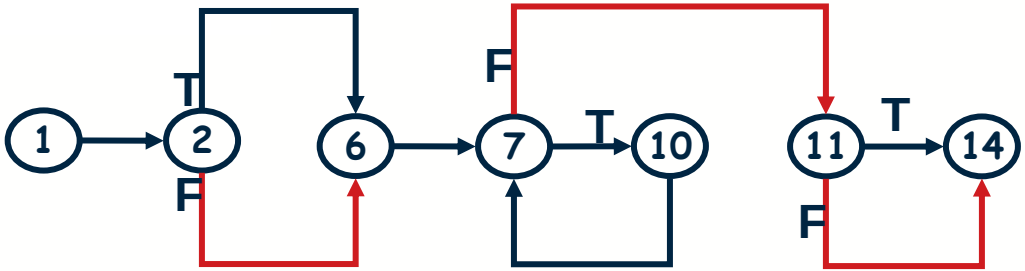
- (x,y)– [(0,0)]

Solution 3b: Branch Coverage

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

For test case (x,y)-[(0,0)]

- Branch covered: 3
- Branch coverage = 3/6



Total number of branches: 6

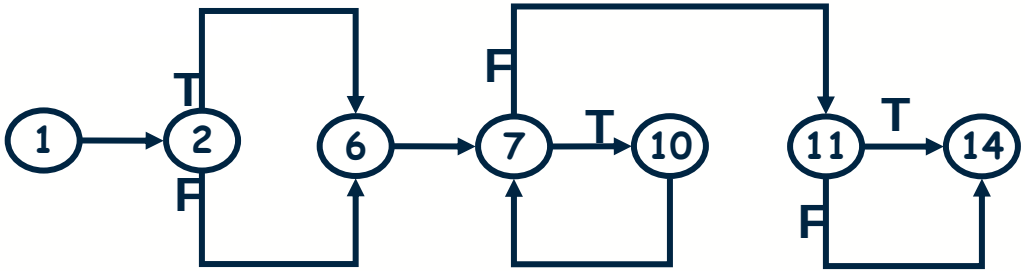
Solution 3b: Decision Coverage

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

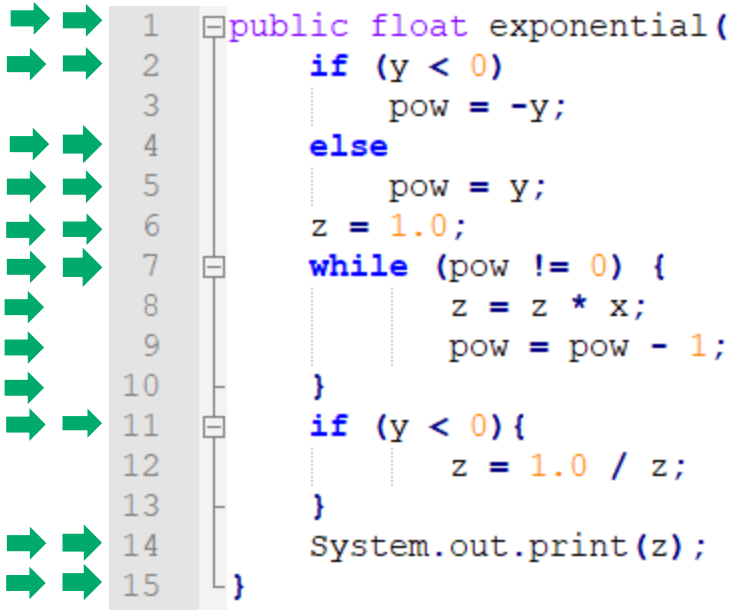
For test case (x,y)-[(0,0)]

- **Decisions covered: 0**
- **Decision coverage = 0/3**

Total number of decisions: 3



Solution 3b: Statement Coverage



```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

For test case (x,y)-[(0,0),(0,1)]

Number of statements covered: 12

Statement coverage = 12/15

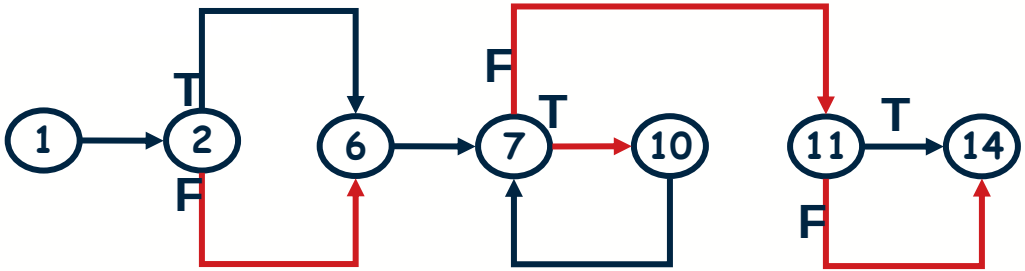
Total number of statements: 15

Solution 3b: Branch Coverage

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

For test case (x,y)-[(0,0),(0,1)]

- Branch covered: 4
- Branch coverage = 4/6



Total number of branches: 6

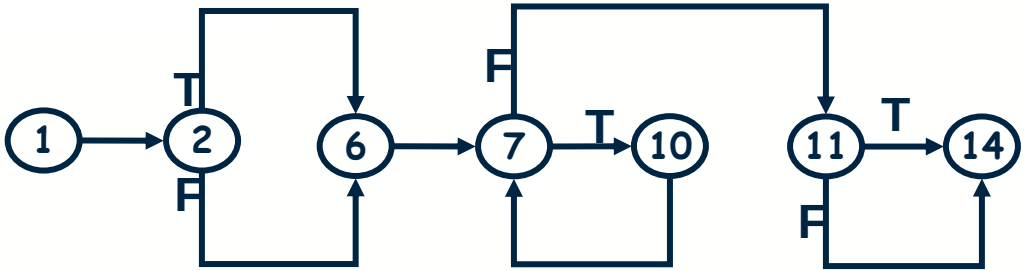
Solution 3b: Decision Coverage

```
1  public float exponential(float x, float y){
2      if (y < 0)
3          pow = -y;
4      else
5          pow = y;
6      z = 1.0;
7      while (pow != 0) {
8          z = z * x;
9          pow = pow - 1;
10     }
11     if (y < 0){
12         z = 1.0 / z;
13     }
14     System.out.print(z);
15 }
```

For test case (x,y)-[(0,0),(0,1)]

- **Decisions covered: 1**
- **Decision coverage = 1/3**

Total number of decisions: 3



Exercise 4: Control Flow Graph

```
1   for (j=1; j<N; j++)  {
2       last = N - j + 1;
3       for (k=1; k<last; k++)  {
4           if (list[k] > list[k+1])  {
5               temp = list[k];
6               list[k] = list[k+1];
7               list[k+1] = temp;
8           }
9       }
10  }
11  System.out.print("Done.");
```

Bubble Sort Algorithm

- What is the Control Flow Graph that describes the above program?

Solution 4: Control Flow Graph