Metamarphic Relations

We define some notations that are used in the description of the MRs, as follows:

- stc: the source test case which includes a Json file and a Java bean
- ftc: the follow-up test case which includes a Json file and a Java bean
- sj: the Json file of source test case
- sb: the Java bean of source test case
- fj: the Json file of follow-up test case
- fb: the Java bean of follow-up test case

Details of each MR are as follows:

1. MR1: fj is obtained by adding a comment at the end of each key-value pair in sj (An example is described below), and sb and fb should have exactly the same content (Member Variables). This MR is violated if a variable with the same name has different values in sb and fb.

```
sb:
public class Test1{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

fb:
public class Test2{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

sj:
{
  "airClass": 2,
  "area": 0,
  "economicfee": 5153.6,
  "luggage": 22.2,
```

```
  "student": false
}

fj:
{
  "airClass": 2,/*this is a comment*/
  "area": 0,/*this is a comment*/
  "economicfee": 5153.6,/*this is a comment*/
  "luggage": 22.2,/*this is a comment*/
  "student": false/*this is a comment*/

}
```

2. MR2: fj is a byte array that is obtained by converting sj into bytes, and sb and fb should have exactly the same content (Member Variables). This MR is violated if a variable with the same name has different values in sb and fb.
3. MR3: fj is obtained by converting sj to a string (An example is described below), and sb and fb should have exactly the same content (Member Variables). This MR is violated if a variable with the same name has different values in sb and fb.

```
sb:
public class Test1{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

fb:
public class Test1{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

sj:
{
  "1": {
    "airClass": 2,
    "area": 0,
    "economicfee": 5153.6,
    "luggage": 22.2,
    "student": false
  }
```

```
}

fj:
\"1\": {\"airClass\": \"2\", \"area\": \"0\", \"economicfee\": \"5153.6\",
\"luggage\": \"22.2\", \"student\": \"false\"}
```

4. MR4: If sj contains a floate-type variable f with the value v, fj sets the value v' (which can be obtained by adding seven significant digits at the end of v) to f , leaving the other variable values unchanged (An example is described below), and sb and fb should have exactly the same content (Member Variables). This MR is violated if the value of f in sb is not equal to v or the value of f variable in fb is not equal to v'.

```
sb:
public class Test1{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

fb:
public class Test1{
  public int airClass;
  public int area;
  public float economicfee;
  public float luggage;
  public boolean student;
  //getter and setter
}

sj:
{
  "1": {
    "airClass": 2,
    "area": 0,
    "economicfee": 5153.6, /*float*/
    "luggage": 22.2,  /*float*/
    "student": false
  }
}

fj:
{
  "1": {
    "airClass": 2,
    "area": 0,
    "economicfee": 5153.61234567,
    "luggage": 22.21234567,
    "student": false
  }
```

```
}
```

5. MR5: If sj contains a double-type variable f with the value v, fj sets the value v' (which can be obtained by adding sixteen significant digits at the end of v) to f , leaving the other variable values unchanged (An example is described below), and sb and fb should have exactly the same content (Member Variables). This MR is violated if the value of f in sb is not equal to v or the value of f variable in fb is not equal to v'.

```
sb:
public class Test1{
  public int airClass;
  public int area;
  public double economicfee;
  public double luggage;
  public boolean student;
  //getter and setter
}

fb:
public class Test1{
  public int airClass;
  public int area;
  public double economicfee;
  public double luggage;
  public boolean student;
  //getter and setter
}

sj:
{
  "1": {
    "airClass": 2,
    "area": 0,
    "economicfee": 5153.6,
    "luggage": 22.2,
    "student": false
  }
}

fj:
{
  "1": {
    "airClass": 2,
    "area": 0,
    "economicfee": 5153.61234567891234567,
    "luggage": 22.2,
    "student": false
  }
}
```

6. MR6: Suppose sb has a enum-type variable $e \in E = \{x_1, x_2, ..., x_n\}$, and the value of e is x_i (i = 1, 2, ... , n) in sj. The value x' of e in the fj is not belong to E (An example is described below), and sb and fb should have exactly the same content (Member Variables). This MR is hold if the value of e in sb is "null" and others values of variables in sb is equal to fb.

```
sb:
enum Code {
  SUCCESS, FAILURE
}

public class Num {
  private int id;
  private Code code;
  //getter and setter
}

fb:
enum Code {
  SUCCESS, FAILURE
}

public class Num {
  private int id;
  private Code code;
  //getter and setter
}

sj:
{
  "code":SUCCESS
  "id":1
}

fj:
{
  "code":ERROR
  "id":1
}
```

7. MR7: Suppose sj has a date-type variable d. fj is obtained by converting the value of d to another date format, and sb and fb should have exactly the same content (Member Variables). This MR is hold if the values of same variables in sb and fb are equal.

```
sb:
public class Test1 {
    private int ID;
    @JSONField(format="yyyy-MM-dd HH:mm:ss")
    private Date date;
    //getter and setter
}
```

```
fb:
public class Test1 {
    private int ID;
    @JSONField(format="yyyy-MM-dd HH:mm:ss")
    private Date date;
    //getter and setter
}

sj:
{
  ID:1,
  date:2018-12-02 13:43:00
}

fj:
{
  ID:1,
  date:13:43:00 2018-12-02
}
```

8. MR8: Suppose sj has a list-type variable *L*. fj is obtained by reversing the order of the elements in the *L* of sj , and sb and fb should have exactly the same content (Member Variables). This MR is hold if the elements of *L* in fb are in reverse order to those in sb, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private List<String> list;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private List<String> list;
    //getter and setter
}

sj:
{
  ID:1,
  list:[
    "apple",
    "banana"
  ]
}

fj:
{
  ID:1,
```

```
  list:[
    "banana",
    "apple"
  ]
}
```

9. MR9: Suppose sj has a list-type variable *L*. fj is obtained by adding an element *x* in the *L* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if the *L* in sb is equal to the *L* in fb by adding *x*, and the values of others variables are equal.

```
sb:
public class Test1 {
    private List<String> list;
    //getter and setter
}

fb:
public class Test1 {
    private List<String> list;
    //getter and setter
}

sj:
{
  list:[
    "apple"
  ]
}

fj:
{
  list:[
    "apple",
    "banana"
  ]
}
```

10. MR10: Suppose sj has a list-type variable *L*. fj is obtained by deleting an element in the *L* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if *L* in fb is missing the deleted element than *L* in sb, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private List<String> list;
    //getter and setter
}
```

```
fb:
public class Test1 {
    private int ID;
    private List<String> list;
    //getter and setter
}

sj:
{
  ID:1,
  list:[
    "apple",
    "banana"
  ]
}

fj:
{
  ID:1,
  list:[
    "apple"
  ]
}
```

11. MR11: Suppose sj has a list-type variable L. fj is obtained by dividing L into two sublists L_1 and L_2. This MR is hold if L_1 and L_2 are combined to obtained L', which is equal to L, and the other contents of fj are equal to sj.

```
sb:
public class Test1 {
    private int ID;
    private List<String> list;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private List<String> list1;
    private List<String> list1;
    //getter and setter
}

sj:
{
  ID:1,
  list:[
    "apple",
    "banana"
```

```
  ]
}

fj:
{
  ID:1,
  list1:[
    "apple"
  ],
  list2:[
    "banana"
  ]
}
```

12. MR12: Suppose sj has a map-type variable *M*. fj is obtained by adding a key-value pair element *x* in the *M* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if the *M* in sb is equal to the *M* in fb by adding the same key-value pair, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private Map map;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Map map;
    //getter and setter
}

sj:
{
  ID:1,
  map:{1:"apple",2:"banana"}
}

fj:
{
  ID:1,
  map:{1:"apple",2:"banana",3:"grape"}
}
```

13. MR13: Suppose sj has a map-type variable *M*. fj is obtained by deleting a key-value pair element in the *M* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if *M* in fb is missing the deleted element than *M* in sb, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private Map map;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Map map;
    //getter and setter
}

sj:
{
  ID:1,
  map:{1:"apple",2:"banana"}
}

fj:
{
  ID:1,
  map:{1:"apple"}
}
```

14. MR14: Suppose sj has a map-type variable *M*. fj is obtained by dividing *M* into two submaps M_1 and M_2. This MR is hold if M_1 and M_2 are combined to obtained *M'*, which is equal to L, and the other contents of fj are equal to sj.

```
sb:
public class Test1 {
    private int ID;
    private Map list;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Map map1;
    private Map map2;
    //getter and setter
}

sj:
{
  ID:1,
  map:{1:"apple",2:"banana"}
```

```
}

fj:
{
  ID:1,
  map1:{1:"apple"},
  map2:{2:"banana"}
}
```

15. MR15: Suppose sj has a set-type variable *S*. fj is obtained by adding an element *x* in the *S* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if the *S* in sb is equal to the *S* in fb by adding *x*, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private Set<String> set;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Set<String> set;
    //getter and setter
}

sj:
{
  ID:1,
  set:["banana","apple"]
}

fj:
{
  ID:1,
  set:["banana","apple"]
}
```

16. MR16: Suppose sj has a set-type variable *S*. fj is obtained by deleting an element *x* in the *S* of sj, and sb and fb should have exactly the same content (Member Variables). This MR is hold if *S* in fb is missing the deleted element than *S* in sb, and the values of others variables are equal.

```
sb:
public class Test1 {
    private int ID;
    private Set<String> set;
```

```
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Set<String> set;
    //getter and setter
}

sj:
{
  ID:1,
  set:["banana","apple"]
}

fj:
{
  ID:1,
  set:["banana"]
}
```

17. MR17: Suppose sj has a set-type variable *S*. fj is obtained by dividing *S* into two subsets S_1 and S_2. This MR is hold if S_1 and S_2 are combined to obtained *S'*, which is equal to *S*, and the other contents of fj are equal to sj.

```
sb:
public class Test1 {
    private int ID;
    private Set<String> set;
    //getter and setter
}

fb:
public class Test1 {
    private int ID;
    private Set<String> set1;
    private Set<String> set2;
    //getter and setter
}

sj:
{
  ID:1,
  set:["banana","apple"]
}

fj:
{
  ID:1,
```

```
  set1:["banana"],
  set2:["apple"]
}
```