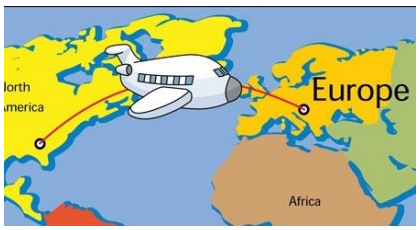
The project is aimed to be useful for java developer. The most obvious use case of this utility is simplification of troubleshooting process and code investigation in places where regular debugging is impossible or problematic:

- QA/system tests etc environments:

- running code is too far and networking overhead becomes significant:



- reproducing complicated scenario: for example, just to check ***smsEngine*.sendMessage()** tons of irrelevant configurations required:

```
void alert(String customerId, String storeId, String productId)
{
  Customer customer;
  Store store;
  if (  (customer=customerRepository.getCustomer(customerId)) != null
        &&  productVerifier.isValid(productId, customer)
        && (store=storeRepository.getStore(storeId)) != null
        &&  zoneService.isSameArea(customer, store)
        &&  discountEngine.getDiscount(customer, store, productId) > 0 ) {
          smsEngine.sendMessage(customer,
                                "Don't miss! we have something interesting..");
  } // if
} // alert
```

- “fast and dirty” workaround for problems which can’t be fixed currently but blocks working on something else:

```
void ourMethod()
{
  //// something
  if ( !valid() )
     throw new IllegalArgumentException("Ooops..");
  //// something
}

boolean valid()
{
  return false;
}
```

- or just library source code is not available but still must be investigated in different cases

Without modification/recompilation of applicative code this utility allows configuration-based non-intrusive monitoring and manipulation of remote jvm. Upon specified event type (arriving to specific source code line, method entry/exit, throwing exception or field modification) it performs required actions, as:

- collecting and logging to console or file information of:
  - variable value
  - all visible local variables values
  - evaluated expression value
  - method arguments
  - method return value
  - stacktrace
  - old/new values in case of field value modification

  etc

- assigning value to variables where value may by
  - constant value
  - value of another variable
  - result of evaluated expression
  - new instance from arbitrary constructor signature

- arbitrary method invocation

- enforcing return from currently executed method with configured return value:

```
boolean valid()
{
  System.out.println("so bad luck..");
  return false; ⇐ without code changing returned value may be enforced to true
}
```

All specified may be performed in conditional way (*equals(..,..), less/greater, notnull, nested or(...) / and(...)*)

The utility is based on java debug interface and no application bytecode modifications involved, as a result when it disconnects from monitored application the last returns to its initial state.

Various configuration examples may be found in /xryusha/onlinedebug/testcases/**.xml files, detailed configuration elements explanation are provided in example_breakpoints.xml, example_actions.xml, example_rvalues.xml and example_condition.xml.