Hacettepe University
Department of Computer Engineering
Bil138 Programming Laboratory

Subject : OOP (Object Oriented Programming)
Programming Language : Java
Submission Date : 25/03/2011
Deadline : 8/4/2011
Advisors : R.A. Levent Seçkin, Dr. Sevil Şen, Dr. Erkut Erdem
Experiment number : 1

AIM

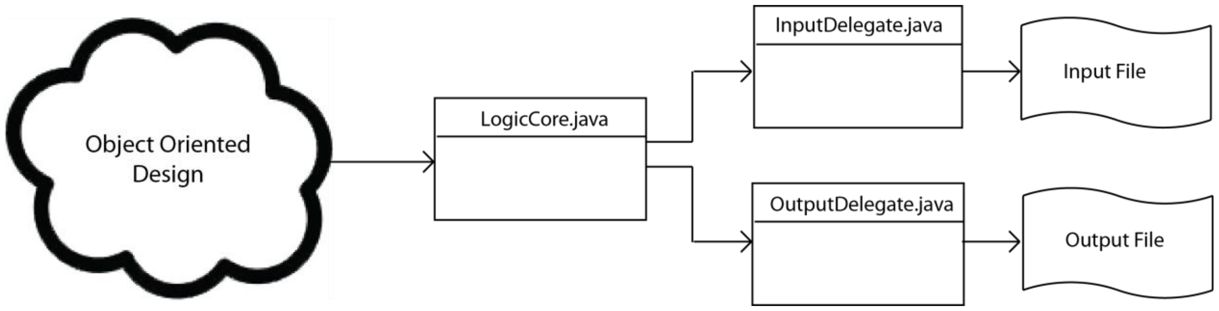
In this experiment you are aimed to get familiar with Object Oriented Programming.You will execute defined commands that are taken from a command file and store the result to output file. While doing this you have to develop a **Real Time Strategy Game Core** in **OOP**.

Object-oriented programming *(**OOP**) is a programming paradigm using "objects" –data structures consisting of data fields and methods together with their interactions – to design applications and computer programs.*[wikipedia]

Experiment

For this experiment you are expected to develop a simple strategy game core. In strategy games more than one players connect to game, they can gather some resources also they can build buildings, create soldiers and they can research on some updates. Your application will take input commands from an input file. Full path of the input file will be given as program argument. Your application prints the outputs of the commands into an output file. Names of these files will be given as argument in the following order: <input_file> <output_file>. In other words your program will be executed as: java LogicCore.class <input_file> <output_file>. Some constants -like prices- and update values will be stored by a pre-defined java class, which will be given to you by this experiment sheet.

Your application must have full Object Oriented Design and all your different data structures must be defined as a class in a remarkable design. General design of your application have to be suitable with this design:



The cloud named 'Object Oriented Design' will be implemented by yourself. This design is the most important part of your program. You have to submit your design in report using the notation above.

Your application should support the following features:

CONNECT PLAYER “<player_name>”
By this command you need to create a player.
output:
<player_name> NAMED PLAYER CREATED
errors:
<player_name> NAMED PLAYER ALREADY CREATED

PLAYER “<player_name>” GATHERED <amount> GOLD
Players can gather gold resource at any time. You need to add defined amount of gold to total resource.
output:
<player_name> NAMED PLAYER GATHERED <amount> GOLD. TOTAL GOLD: <amount>
errors:
<player_name> NAMED PLAYER CAN NOT FOUND

PLAYER “<player_name>” CREATE <type> "<id>"
Players can create farm and soldier types. Soldiers and farms will have a unique ID to reach them when needed. When a soldier is created you have to check available slot of player, creating a soldier type will decrease available slot and building a farm will increase number of available slots.(amount of increase is defined at constant class). Creating any type of new item causes decrease in gold resource. Prices are defined at constant class.
<type> := FARM, SOLDIER, HORSEMAN, CATAPULT
output:
<id> NAMED <type> CREATED FOR PLAYER <player_name>
errors:
<player_name> NAMED PLAYER CAN NOT FOUND
NOT ENOUGH SLOT TO CREATE <type>
NOT ENOUGH GOLD TO CREATE <type>

PLAYER “<player_name>” REMOVE <type> "<id>"
Any item can remove from system at any time.
<type> := FARM, SOLDIER, HORSEMAN, CATAPULT
output:
<id> NAMED <type> REMOVED FOR PLAYER <player_name>
errors:
<player_name> NAMED PLAYER CAN NOT FOUND
<id> NAMED <type> CAN NOT FOUND

PLAYER “<player_name>” UPGRADE <soldier_type> <upgrade_type>
This command doesn’t affect already created soldier. Only the soldiers which will be created *after this command* will be effected. Prices are defined at constant class.
<soldier_type> := SOLDIER, HORSEMAN, CATAPULT
<upgrade_type> := ATTACK, DEFENCE

output:
<soldier_type> <upgrade_type> UPGRADED FOR PLAYER <player_name>
errors:
 <player_name> NAMED PLAYER CAN NOT FOUND
NOT ENOUGH GOLD TO UPDATE <soldier_type> <upgrade_type>

PLAYER “<player_name>” UPGRADE FARM “<id>”
Upgrading a farm increases available slots, which is defined at constant class, of this farm. Prices are defined at constant class.
output:
<id> NAMED FARM UPGRADED FOR PLAYER <player_name>
errors:
 <player_name> NAMED PLAYER CAN NOT FOUND
<id> NAMED FARM CAN NOT FOUND
NOT ENOUGH GOLD TO UPDATE FARM

STATUS PLAYER “<player_name>” <type> “<id>”
When you process this command you need to give information about current status of this item.
< type> := GENERAL, FARM, SOLDIER, HORSEMAN, CATAPULT
output general:
PLAYER NAME:<player_name>
AVAILABLE GOLD: *<amount>*
AVAILABLE SLOT: *<amount>*
TOTAL FARM COUNT: *<amount>*
TOTAL SOLDIER COUNT: *<amount>*
TOTAL HORSEMEN COUNT: *<amount>*
TOTAL CATAPULT COUNT: *<amount>*

output farm:
FARM ID: <id>
SLOT: *<amount>*

output soldier types:
<soldier_type> ID: <id>
HEALTH: <amount>
ATTACK FORCE: <amount>
DEFENCE FORCE: <amount>

errors:
 <player_name> NAMED PLAYER CAN NOT FOUND
<id> NAMED <type> CAN NOT FOUND

ATTACK PLAYER “<player_name_attack>” < soldier_type_attack> "<id_attack>" PLAYER “<player_name_DEFENCE>” < soldier_type_ DEFENCE > "<id_ DEFENCE >"
This command is bonus and will grant you extra credit if you implement this feature.

You need to get your constant values from a pre-defined class. This class must be declared as:

```
public class StrategyConstants {

      public static final int PLAYER_START_GOLD = 0;
      public static final int PLAYER_START_FARM_SLOT = 0;

      public static final int FARM_START_SLOT_CAPACITY = 6;
      public static final int FARM_CREATE_COST = 50;
      public static final int FARM_UPGRADE_SLOT = 3;
      public static final int FARM_UPGRADE_COST = 23;

      public static final int SOLDIER_CREATE_COST = 100;
      public static final int SOLDIER_SLOT_NEED = 1;
      public static final int SOLDIER_START_HEALTH = 50;
      public static final int SOLDIER_START_ATTACK = 5;
      public static final int SOLDIER_START_DEFENCE = 3;

      public static final int HORSEMAN_CREATE_COST = 150;
      public static final int HORSEMAN_SLOT_NEED = 3;
      public static final int HORSEMAN_START_HEALTH = 150;
      public static final int HORSEMAN_START_ATTACK = 5;
      public static final int HORSEMAN_START_DEFENCE = 3;

      public static final int CATAPULT_CREATE_COST = 450;
      public static final int CATAPULT_SLOT_NEED = 6;
      public static final int CATAPULT_START_HEALTH = 50;
      public static final int CATAPULT_START_ATTACK = 25;
      public static final int CATAPULT_START_DEFENCE = 3;

      public static final int ATTACK_DEFENCE_UPGRADE = 3;
      public static final int ATTACK_DEFENCE_UPGRADE_COST = 50;

}
```

Important Notes

- You have to design your application in complete object oriented design approach. You are expected to define a java class for everything that is suitable for a class.
- Class diagram for full program must be given in your report. You have prepare a basic class diagram so you need to do little research about UML Class Diagrams.
- Don't make any change on constants class.
- Another important point is designing your code in encapsulation paradigm. Encapsulation is an obligation in OOP approach.
- You will use *minio* library for IO operations. You can download this library from: http://web.cs.hacettepe.edu.tr/~ersiner/prj/minio/ . Also you can find the usage documentary from this link.
- ATTACK is a bonus command. You don't have to implement this feature. Students who implement this feature will be rewarded by extra credit.
- Example input and output files will be published. Checking your code with them can help you.

- The inventory will be filled using data in the file of which format is given above. Note that input file may include whitespace (space or tab) characters. Just remove extra whitespaces, leaving only one space character between fields. There can be empty lines in the input file.
- In the output file there should be a separator consisting of 10 dashes between each command output.

SUBMISSION

- Your submission will be in the format below

```
 < StudentID>
  |-- report
report.pdf
  |-- source
LogicCore.java
InputDelegate.java
OutputDelegate.java
StrategyConstants.java
other  *.java files
```

- You have to use “Online Experiment Submission System”.
- http://submit.cs.hacettepe.edu.tr. Other type of submissions especially by e-mail WILL NOT BE ACCEPTED.
- Submission deadline is 8/4/2011, 16.59 pm.
- Do not forget to sign the "submission paper" after you submit your experiment.
- Respect the office hours of your advisor .

  Wednesday 09:00-12:00, Friday 13:00-15:00 Office: 122

REFERENCES

- http://www.google.com
- http://en.wikipedia.org/wiki/Object-oriented_programming
- http://en.wikipedia.org/wiki/Real_time_strategy_game
- http://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)
- http://web.cs.hacettepe.edu.tr/~ersiner/prj/minio/
- http://en.wikipedia.org/wiki/UML
- http://en.wikipedia.org/wiki/Class_diagram