Michae Hiszpanski

Demonstrate a knowledge of fundamental object-oriented programming constructs appropriate to the selected language by writing and implementing code applicable to a given specification 2 Demonstrate an object-oriented approach to design through creation of appropriate design documentation for a given specification 3 Facilitate readability and maintainability of a solution through design and development of code to a professional standard 4 Determine the correctness of a solution through creation and application of appropriate test strategies

Level 5 Foundation Degree in Science in Computing

Module : OOP Assignment2 :Explore!

Table of Contents

[Introduction 1](#_Toc71292828)

[Pseudo Code 2](#_Toc71292829)

[Task 1 & 2 3](#_Toc71292830)

[RoW class 3](#_Toc71292831)

[Hazard class 4](#_Toc71292832)

[Winter class 4](#_Toc71292833)

[Environment class 5](#_Toc71292834)

[Code changes 6](#_Toc71292835)

[Example of Interface 6](#_Toc71292836)

[GUI class 7](#_Toc71292837)

[RedirectingOutputStream & ConsoleToGUI 7](#_Toc71292838)

[Tests - Task 3 8](#_Toc71292839)

[Test 1 8](#_Toc71292840)

[Test 2 date problem … 11](#_Toc71292841)

[Whitebox test 14](#_Toc71292842)

[Error check test 17](#_Toc71292843)

[Last Test 19](#_Toc71292844)

[Conclusion 22](#_Toc71292845)

[Referencing 22](#_Toc71292846)

# Introduction

Project 'Explore!' set new challenges and difficulties. I had to start over several times and finally the current version of the program was created, which can be called the Vacation Plan. I tried to have a fully working Interface using JFrame. Problems with notation and date format forced me to limit myself to the presentation version. The program consists of 2 classes (HolidayPlan, Policy) 3 subclass (Row, Hazard, Winter), Environment and interface builder classes (GUI, RedirectingOutputStream, ConsoleToGUI). Environment class is the place of relationships between classes and subclasses and the order of the program's actions is written there. The program starts with creating an Environment class object in the main method. This is what the ‘MainStart’ class is for. The code is run with one line of code.

Please use ‘MainStart’ class to start program.

The GUI is the class that creates the final interface. This is a small presentation of my possibilities. After filling the bar on the panel 'panel' on textArea the output in String format will be displayed. After calling the GUI, everything on the console will be displayed in textArea. It is enough to call toString from the HolidayPlan and Policy classes, as well as the DisplayTravellers method from the appropriate class selected by the user.

'Click ME Last !!' button in the 'panel2' panel it is inactive until the bar is full. The user can then enter any String value in the text field also in ‘panel2’. After pressing the 'button' button, the 'panel2' panel becomes invisible and its contents. In its place appears the 'panel3' panel with labels 'label' and 'label2'. They were arranged in any way. The third element of the 'panel3' panel is the button (button2) 'Click to Exit !! '. ‘button2’ exits the program after clicking the mouse. Additional information is that the value of textField entered by user also displays in the textArea in the GUI.

# Pseudo Code

First step

|  |
| --- |
| * Creating HolidayPLan object * Input ID: String * Input Holiday Plan owner name: String * Input destination: String * Input number of Travellers: int * Input ame travellers :ArrayList<String> * Input DoB Travellers: ArrayList<Calendar> * Choose Policy Type |

Second step

|  |  |  |
| --- | --- | --- |
| RoW | Hazard | Winter |
| * Input Policy number: String * Input Policy holder name: String * Input holder DoB: Calendar * Input start date :Calendar * Input end Date Calendar * Choose part of Word going to :String * Display Holiday Plan and Policy information * Make user choose to accept or cancel Policy * Display Holiday Plan and Policy information | * Input Policy number: String * Input Policy holder name: String * Input holder DoB: Calendar * Input start date :Calendar * Input end Date Calendar * Choose part of Word going to :String * Choose user Activity : ArrayList * Display Holiday Plan and Policy information * Make user choose to accept or cancel Policy * Display Holiday Plan and Policy information | * Input Policy number: String * Input Policy holder name: String * Input holder DoB: Calendar * Input start date :Calendar * Input end Date Calendar * Choose part of Word going to :String * Choose user Activity : ArrayList * Choose user Level: ArrayList * Display Holiday Plan and Policy information * Make user choose to accept or cancel Policy * Display Holiday Plan and Policy information |

Step three

|  |
| --- |
| * Start and display information in the GUI * small GUI presentation for user * end program |

# Task 1 & 2

## RoW class

|  |
| --- |
| RoW(Extends Policy) |
| #Destinations:Arraylist<String>  #userDestination:String |
| +RoW  +RoW(HolidayPlan)  +setDestinations(String):void  +getDestinations: Arraylist<String>  +getUserDestination :String  + setUserDestination(String):void  +toString :String  +DisplayTravellers :String  +calPremium :void  +addDestinations :void  +chooseDestinations :void  +chooseAndsetDailyRate :double  +keyboardInputInteger (int,int) :int |

RoW 'Rest of World' class. In this class, dailyRate is calculated based on the user's choice. User selects one of 4. Here we have the keyboardInputInteger method, thanks to which the user, if he wants to continue working with the program, must select an integer from a given range. There is no other possibility. This method has largely solved the problem of entering appropriate values ​​in dates.

## Hazard class

|  |
| --- |
| Hazard (Extends RoW) |
| #Activity:ArrayList<String>  #TravellerActivity:ArrayList<String> |
| +Hazard  +Hazard(HolidayPlan)  +setActivity(String): :ArrayList<String>  +setTravellerActivity(String) :ArrayList<String>  +getActivity: :ArrayList<String>  +getTravellerActivity:ArrayList<String>  +userDestinations:void  +addActivity:void  +makeChoose:void  +chooseActivity (HolidayPlan) :void  +calPremium:void  +toString :String  +DisplayTravellers (HolidayPlan) :String |

The 'Hazard' class will develop the 'RoW' class. In the Hazard class, dailyRate has a predetermined value, but the user still chooses which part of the world to go to (option 1 of 4) and additionally selects the type of activity (option 1 of 5) for individual travelers. I added three additional activities (Skates, Hiking and Climbing ) to diversify both the choice and the program.

## Winter class

|  |
| --- |
| Winter (Extends Hazard) |
| #Levels:ArrayList<String>  #TravellerLevels:ArrayList<String>  #EquipmnetCover: final double  #highestRate:double |
| +Winter  +Winter (HolidayPlan)  +getHighetsRate:double  +setHighetsRate(double):void  +getTravellerLevel: ArrayList<String>  +setTravellerLevel(String): ArrayList<String>  +addLevels:void  +chooseLevel:void  +userChooseLevel:void  +chooseActivity(HolidayPlan):void  +setproperDailyRate(HolidayPlan):double  +calPremium:void  +calPremium(HolidayPlan):void  +toString:String  +DisplayTravellers (HolidayPlan) :String |

The 'Winter' class is an extension of the 'Hazard' class. It is the most extensive subb class of the super Policy class. DailyRate (its value for a particular day) is calculated on the basis of the type of sports activity selected by travelers. The highest value is used as the one-day cost for each traveler. Additional cost in the form of equipment fee for each traveler. Based on the sum of these activities above, we have calculated the total cost of the policy.

## Environment class

|  |
| --- |
| Environment |
| #userPolicy:ArrayList<Policy>  #policy:Policy  #plan:HolidayPlan  #endpolicydate:Calendar |
| +Environment  +getPolicy:Policy  +setPolicy(Policy):void  +getUserPolicy:ArrayList<Policy>  +setUserPolicy(Policy):Policy  +getPlan:HolidayPlan  +setPlan:void  +getEndPolicydate:Calendar  +setEndPolicyDate(Calendar):void  +getRoW:RoW  +getHazard:Hazard  +getWinter:Winter  +addingObjects:void  +choosePolicy:int  +userPolicy:Policy  +Action:void  +Cancel:void |

The Environment class is the class that corresponds to the order of relations, interactions between particular classes. It was written to allow the program to start by calling instances of the Environment class. An additional method, an option in this class is the Cancel method. Thanks to this method, the user has the opportunity to decide whether he accepts the Policy and the costs, or resigns from this Policy. The information displayed at the end is appropriately corrected to confirm the user's choice.

## Code changes

|  |
| --- |
| HolidayPlan |
| none |
| +Input:void  +addTravelers:void  +outputTravelers:void  +addtravellersDoB:void  +setAnYDate:Calendar  +checkuserInputDate:Calendar  +makeDateLimited |

In the Holiday Plan class, the methods that are shown in the diagram above have been added to the original code. The changes were made by adjusting the class itself to the whole project. Additionally, the changes were made in the constructor. ArrayList ‘Travelers’ and ‘dod’ ArrayList-Callendar are initialized at the time of association. The toString method has been slightly modified. Comments in the code should explain the use and construction of additional methods There is a problem with hanging the date in the right format. The makeDateLimited method fixes the problem with entering the appropriate values ​​for days, months or years, and the date itself is returned as a String value, which is then converted into a Calendar variable with the setAnYDate method.

|  |
| --- |
| Policy |
| +Policy(HolidayPlan) |
| +inputPolicy:void  +checkEndDate:void  +currentUserInputDate:Calendar  +DisplayTravellers(HolidayPlan):String |

A new constructor with HolidayPlan has been added as a parameter in the Policy class. Certain information will be requested in the Policy or its subclasses and will not be released until the HolidayPlan is established. The methods shown above in the diagram are those that were added to the original code. Comments in the code should explain the operation and sense of the solutions used.

## Example of Interface

### GUI class

|  |
| --- |
| GUI(Extends JFrame Implements ActionListener) |
| -textArea:JTextArea  -frame:JFrame  -panel,panel2,panel3:JPanel  -bar:JProgressBar  -button,button2:JButton  -textfield:TextField  -text:String |
| +GUI  +getPanels  +getTextArea  +getBar  +getButton  +getLabel  +getTextField  +setText(String):void  +getText:String  +toString:String  +actionPerormed:void @Override  +fill:void  +start:void  +appendText:void |

GUI classes is a demo interface. Its task is to show you the possibility of using JFrame. Thanks to the implementation of ActionListener, we can associate individual changes, actions after clicking the appropriate buttons. This is a small demonstration of how to use panels, label, text area, text field, button, progress bar. A detailed description can be found in commentaries. The interface itself is started only when the user makes a choice when calling the Cancel method in the Environment class

### RedirectingOutputStream & ConsoleToGUI

|  |
| --- |
| RedirectingOutputStream( Extends OutputStream) |
| -gui:GUI |
| +write(int):void @Override |

|  |
| --- |
| ConsoleToGUI |
| none |
| +ConsoleToGUI  +run:void -- not used |

Java IO is an API (Application Programming Interface) that comes with Java which is targeted at reading and writing data (input and output). For instance, read data from a file or over network, and write to a file or write a response back over the network. The Java IO API is located in the Java IO package. Java IO basically provides a mechanism to read data from a source and write data to a destination. Input represents the source while output represents the destination here. These sources and destinations can be anything from Files, Pipes to Network Connections. Java IO provides the concept of streams which basically represents a continuous flow of data. Streams can support many different types of data like bytes, characters, objects, etc. It has been used to call the output from console on textArea.

# Tests - Task 3

## Test 1

The first test to check the operation of dates and the program itself. Thanks to the 'makeDateLimited' method in the HolidayPlan class, the program forces the user to enter appropriate values ​​for the day, month and year separately. In this test, the Hazard policy was chosen and finally accepted

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HolidayPlan | | Default | | Input | | Output | |
| id | | “ “ | | 0007GH | | 0007GH | |
| name | | “ “ | | Michael H | | Michael H | |
| dateCreated | | Current date | |  | | 5/5/2021 | |
| policy | | null | | Hazard | | Assignment2OOp.Hazard | |
| insuranceStatus | | “client has own” | |  | | 000HGF12 | |
| destination | | “” | | Italy | | Italy | |
| Cost | | 0 | |  | | 28.00 | |
| Travellers | Name | | DoB | |  | |  |
| 1 | Trav1 | | 2/2/1982 | |  | |  |
| 2 | Trav2 | | 12/3/1984 | |  | |  |
| 3 | Trav3 | | 12/5/1984 | |  | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Policy(Hazard) | | Default | | Input | | Output | |
| policyNum | | “ “ | | 000HGF12 | | 000HGF12 | |
| name | | “ “ | | Lisa | | Lisa | |
| status | | “pending” | |  | |  | |
| dob | | null | | 12/2/1991 | | 12/2/1991 | |
| sDate | | null | | 15/5/2022 | | 15/5/2022 | |
| eDate | | null | | 17/5/2022 | | 17/5/2022 | |
| dailyRate | | 0 | |  | | 14 | |
| premium | | 0 | |  | | 28 | |
| holidayPlanID | | “” | |  | | 007GH | |
| User Destination | |  | | 4 | | Rest World | |
| Travellers | Name | | DoB | | Activity | | Level |
| 1 | Trav1 | | 2/2/1982 | | Snowboard | |  |
| 2 | Trav2 | | 12/3/1984 | | Skies | |  |
| 3 | Trav3 | | 12/5/1984 | | Skates | |  |

Output Before user make decision .

Text

Description automatically generated

Output after decision was made.

A picture containing shape

Description automatically generatedText

Description automatically generated with medium confidence

Last screen shoot shows output on console and GUI interface

Text

Description automatically generated with medium confidence

After clicking button “click to Exit “ program Exit .

## Test 2 date problem …

In this test makeDateLimited method from HolidayPlan class in not used.

This test is intended to show errors when typing wrong values ​​in the format date. As you can see, premium is not countable because the multipliers are too large. The setAnYDate method works so that it will transfer the integer values ​​between '/' into years. If we enter too many days, the surplus will be converted into months or years. This explains why the results of the annuals are not realistic, reaching many thousands of years. The makeDateLimited method fixes this error to some extent. With this method, restrictions are overridden to prevent strange results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HolidayPlan | | Default | | Input | | Output | |
| id | | “ “ | | ---777 | | ---777 | |
| name | | “ “ | | Michael | | Michael | |
| dateCreated | | Current date | |  | | 5/5/2021 | |
| policy | | null | | 3 | | winter | |
| insuranceStatus | | “client has own” | |  | | 97 | |
| destination | | “” | | Belfast | | Belfast | |
| Cost | | 0 | |  | | 16542545700.00 | |
| Travellers | Name | | DoB | |  | |  |
| 1 | Trav1 | | 3/3/2085 | |  | |  |
| 2 | Trav2 | | 1/1/2737908 | |  | |  |
| 3 | Trav3 | | 9/2/2433696 | |  | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Policy(Winter) | | Default | | Input | | Output | |
| policyNum | | “ “ | | 97 | | 97 | |
| name | | “ “ | | Randomname | | Randomname | |
| status | | “pending” | | CANCELLED | | CANCELLED | |
| dob | | null | | 6666666666/1/1 | | 30/4/1825272 | |
| sDate | | null | | 12/12/2021 | | 12/12/2021 | |
| eDate | | null | | 99999999999999/1/1-first end input date 999999/999999/1- second input end date after cancelled | | To calculate premium date is 21/10/756887  After, last output  8/10/86073 | |
| dailyRate | | 0 | | 5 (Climbing) | | 20.0 | |
| premium | | 0 | |  | | 1.65425457E10 | |
| holidayPlanID | | “” | |  | | ---777 | |
| User Destination | |  | | 1 | | North Amercia | |
| Equipment fee | | 40 | | X3 | | 120 | |
| Travellers | Name | | DoB | | Activity | | Level |
| 1 | Trav1 | | 3/3/2085 | | Climbing | | Master |
| 2 | Trav2 | | 1/1/2737908 | | Hiking | | Advanced |
| 3 | Trav3 | | 9/2/2433696 | | Skates | | Intermediate |

Text

Description automatically generatedText

Description automatically generated

The screen shoot above shows the possibility of entering a huge number in place of days or months and then I will program to replace it with the appropriate date which gives the result of the year often in thousands.

Output before use Cancelled his Polisy.

Text

Description automatically generated

Output before user Click buton ‘Clicl Me Last!!’

Text

Description automatically generated

Output after user click button ‘Click Me Last!!’’ and Before user can choose to close the program between clicking button2 ‘Click to Exit !!’ or the 'x' in the upper right corner of the GUI.

Graphical user interface, text

Description automatically generated

## Whitebox test

The white box is a test of a person who does not know the program structure or even the Java language itself. This person was invited to test the program without having any knowledge about the project itself and the sequence of actions. This test meets the conditions of White Box testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HolidayPlan | | Default | | Input | | Output | |
| id | | “ “ | | 2446 | | 2446 | |
| name | | “ “ | | mateusz | | mateusz | |
| dateCreated | | Current date | |  | | 5/5/2021 | |
| policy | | null | | Winter | | Assignment2OOP.Winter | |
| insuranceStatus | | “client has own” | | “client has own” | | 4 | |
| destination | | “” | | hotel | | hotel | |
| Cost | | 0 | |  | | 18392.00 | |
| Travellers | Name | | DoB | |  | |  |
| 1 | Mateusz | | 27/3/1992 | |  | |  |
| 2 | Agnieszka | | 7/4/1993 | |  | |  |
|  |  | |  | |  | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Policy(Winter) | | Default | | Input | | Output | |
| policyNum | | “ “ | | 4 | | 4 | |
| name | | “ “ | | mateusz | | mateusz | |
| status | | “pending” | |  | | CANCELLED | |
| dob | | null | | 4 3 1996 | | 4/3/1996 | |
| sDate | | null | | 2 4 2023 | | 2/4/2023 | |
| eDate | | null | | Before 4/5/2025 user Cancelled  After 3 4 2024 | | 3/4/2024 | |
| dailyRate | | 0 | |  | | 12 | |
| premium | | 0 | |  | | 18392.0 | |
| holidayPlanID | | “” | |  | | 2446 | |
| User destination | |  | | 2 | | South America | |
| Equipment | | 40 | |  | | 80.0£ | |
| Travellers | Name | | DoB | | Activity | | Level |
| 1 | mateusz | | 27/3/1992 | | Snowboard | | Advanced |
| 2 | agnieszka | | 7/4/1993 | | Snowboard | | Advanced |

Text

Description automatically generated

Premium is calculated from sDate 2/4/2023 to eDate 4/5/2025 increased by the number of travelers. Daily Rate 12. Equipment 80. Total :18392.0£

Text

Description automatically generated

Output after user click button ‘Click Me Last!!’’ and Before user can choose to close the program between clicking button2 ‘Click to Exit !!’ or the 'x' in the upper right corner of the GUI.

Text

Description automatically generated

## Error check test

Test checking for errors in the program, especially paying attention to dates and their dependence.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HolidayPlan | | Default | | Input | | Output | |
| id | | “ “ | | haha1234 | | haha1234 | |
| name | | “ “ | | Harry | | Harry | |
| dateCreated | | Current date | |  | | 5/5/2021 | |
| policy | | null | | 1 | | Assignment2OOP.RoW | |
| insuranceStatus | | “client has own” | |  | | 9789 | |
| destination | | “” | | 3 | | Asia | |
| Cost | | 0 | |  | | -69030.00 | |
| Travellers | Name | | DoB | |  | |  |
| 1 | Trav1 | | 12/2/1987 | |  | |  |
| 2 | Trav2 | | 1/1/2000 | |  | |  |
| 3 | Trav3 | | 3/3/2003 | |  | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Policy(RoW) | | Default | | Input | | Output | |
| policyNum | | “ “ | | 9789 | | 9789 | |
| name | | “ “ | | Michael | | Michael | |
| status | | “pending” | | active | |  | |
| dob | | null | | 16 6 1982 | | 16/6/1982 | |
| sDate | | null | | 1 2 2021 | | 1/2/2021 | |
| eDate | | null | | 2 2 2000 | | 2/2/2000 | |
| dailyRate | | 0 | | 3 | | 9 | |
| premium | | 0 | |  | | -69030.0 | |
| holidayPlanID | | “” | |  | | haha1234 | |
| Travellers | Name | | DoB | | Activity | | Level |
| 1 | Trav1 | | 12/2/1987 | |  | |  |
| 2 | Trav2 | | 1/1/2000 | |  | |  |
| 3 | Trav3 | | 3/3/2003 | |  | |  |

This scren shot below shows that you are trying to enter a value other than what the program requires at the moment.

Text

Description automatically generated

Output, screen shot of GUI and console before user click on button ‘Click me Last!!’ and typed ‘Hello oh and Byesfsfd’ on text field

Text

Description automatically generated

Output after user click button ‘Click Me Last!!’’ and Before user can choose to close the program between clicking button2 ‘Click to Exit !!’ or the 'x' in the upper right corner of the GUI.

Graphical user interface, text

Description automatically generated

End of date error in the policy class that arises when the makeDateLimited method is used in the HolidayPlan class results from the 'while' loop, in method checkEndDate in Policy class, being 'eDate = sDtae;' . For the first time, if we enter the data in a lower year or wrong, the loop will perform the operation correctly once and ask you to enter the end date again. However, the next time it skips the condition in the while loop. Therefore, the result is a minus.

## Last Test

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HolidayPlan | | Default | | Input | | Output | |
| id | | “ “ | | 0007--- | | 0007--- | |
| name | | “ “ | | Michael | | Michael | |
| dateCreated | | Current date | |  | | 7/5/2021 | |
| policy | | null | | 2 | | Hazard | |
| insuranceStatus | | “client has own” | |  | | Policynumber | |
| destination | | “” | | Poland | | Poland | |
| Cost | | 0 | |  | | 140 | |
| Travellers | Name | | DoB | |  | |  |
| 1 | Trav1 | | 1/2/1922 | |  | |  |
| 2 | Trav2 | | 2/2/2000 | |  | |  |
| 3 | Trav3 | | 3/3/1990 | |  | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Policy | | Default | | Input | | Output | |
| policyNum | | “ “ | | Policynumber | | Policynumber | |
| name | | “ “ | | Somename | | Somename | |
| status | | “pending” | |  | | active | |
| dob | | null | | 2 2 1999 | | 2/2/1999 | |
| sDate | | null | | 2 2 2021 | | 2/2/2021 | |
| eDate | | null | | 12 2 2021 | | 12/2/2021 | |
| dailyRate | | 0 | |  | | 14 | |
| premium | | 0 | |  | | 140 | |
| holidayPlanID | | “” | |  | | 0007--- | |
| Travellers | Name | | DoB | | Activity | | Level |
| 1 | Trav1 | | 1/2/1922 | | Hiking | |  |
| 2 | Trav2 | | 2/2/2000 | | Climbing | |  |
| 3 | Trav3 | | 3/3/1990 | | Snowboard | |  |

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

# Conclusion

Entering the date in the correct format turned out to be the biggest challenge. Using the setAnYDate method solves the problem halfway. The solution I introduced in the checkuserInputDate method with the minimum date makes sense when in the setAnyDate method the String 'date' value is written using keyboard.nextLine; . But there is still the problem of entering the date of the year above the current one. You can also replace the days or months with an Integer number so large that it will be converted into years and will meet the condition of the minimum date. Another method that returns String makeDateLimited addresses this problem. You can use the keyboardInputInteger method in the RoW class to write days, months, and years as int. Then convert it to a String in the appropriate format that meets the conditions of the setAnYDate method. Then we use date = makeDtaeLimited instead of date = keyboartd.nextLine. This solution gives interesting possibilities but created the problem of checking the end date to the start date in the selected policy. As a result, the results of cost can be negative. (Look-- Error check Test)

# Referencing

1. Books

“Think Java” – Allen B. Downey & Chris Mayfield. Second edition

“Java- in 24 Hours” – Rogers Cadenhead. Seventh edition

1. Links

[www.stackoverflow.com](http://www.overstackflow.com)

[www.google.com](http://www.google.com)