TMA01-32J-Solutions

# Question 1

a)

1. Code

/\*\*

\* SafeState Class - SafeState Managers the State of Class Safe

\* Enabling Users to Set/Get the current State of a Safe

\* Object

\*

\* @author Joel Manning

\* @version 23-10-2023

\*/

public class SafeState

{

1. CodeRunner ResultsA blue and white striped background

   Description automatically generated

b)

1. Code

/\*\*

\* @brief part(b) private instance fields for SafeState

\*/

private String userCode, masterCode, display;

private boolean open;

1. CodeRunner Results

A blue and white striped background

Description automatically generated

c)

1. Code

/\*\*

\* @brief Constructor for objects of class SafeState

\*/

public SafeState()

{

//(c) Initialise fields to default values

this.userCode = null;

this.masterCode = "9999";

this.display = "WELCOME";

this.open = true;

}

1. CodeRunner Results

A close-up of a computer screen

Description automatically generated

d)

1. Code

/\*\*

\* @brief (d)(part i)Method for retrieving display field value

\*/

public String getDisplay()

{

return this.display;

}

1. CodeRunner Results

A close-up of a code

Description automatically generated

e)

1. Code

/\*\*

\* @brief part(e) Method for opening the safe

\*/

public boolean open(String aCode)

{

if( this.open == true )

{

this.display = "ALREADY OPEN";

}

else if( this.open == false )

{

if( aCode == this.userCode || aCode == this.masterCode )

{

this.open = true;

this.display = "OPEN";

this.userCode = null;

}

else

{

this.display = "INVALID CODE";

}

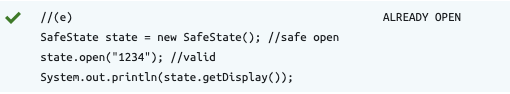
}

// Will return state of open field

return this.open;

}

1. CodeRunner Results



f)

1. Code

/\*\*

\* @brief part(f) Method for checking a valid code for new user code

\* @param String aCode

\*/

public boolean isValidUserCode(String aCode)

{

/\*\*

\* Immutable code length as spec states constant

\*/

final int codeLength = 4;

if(aCode.length() != codeLength){

return false;

}

/\*\*

\* for loop using i as index and charAt method to isolate

\* char for validation check

\*/

for(int i = 0; i < aCode.length(); i++){

if(aCode.charAt(i) != '0' &&

aCode.charAt(i) != '1' &&

aCode.charAt(i) != '2' &&

aCode.charAt(i) != '3' &&

aCode.charAt(i) != '4' &&

aCode.charAt(i) != '5' &&

aCode.charAt(i) != '6' &&

aCode.charAt(i) != '7' &&

aCode.charAt(i) != '8' &&

aCode.charAt(i) != '9')

{

return false;

}

}

return true;

}

1. CodeRunner Results

A screenshot of a computer code

Description automatically generated

g)

1. Code

/\*\*

\* @brief part(g) Method to lock the Safe Providing Correct String has been entered

\* @param String aCode

\*/

public boolean lock (String aCode)

{

boolean lockedState = false;

/\*\*

\* part(g)(part i) Check if Safe is Locked

\*/

if(this.open == false)

{

lockedState = false;

this.display = "ALREADY LOCKED";

}

else

{

/\*\*

\* part (g)(part ii) using internal isValidUserCode method determine

\* is passed code is valid

\*/

if(isValidUserCode(aCode) == true)

{

this.open = false;

this.display = "LOCKED";

this.userCode = aCode;

lockedState = true;

}

/\*\*

\* part (g)(part iii) Set display to `INVALID CODE` if code is invalid

\*/

else

{

this.display = "INVALID CODE";

}

}

return lockedState;

}

1. CodeRunner Results

A screenshot of a computer code

Description automatically generated

h)

1. Code

\*\*

\* @brief part (h) Display the current status of the Safe

\*/

public String about()

{

return "Safe " + this.display;

}

1. CodeRunner Results

A screenshot of a computer code

Description automatically generated

# Question 2

a)

1. Code

/\*\*

\* @brief Java Library Imports

\*/

import java.util.ArrayList;

import java.util.Iterator;

/\*\*

\* Class 'Safe' for containing a number of limited items to be safely stored

\*

\* @author Joel Manning

\* @version 31-10-2023

\*/

public class Safe

{

1. CodeRunner Results

A screenshot of a computer

Description automatically generated

b)

1. Code

/\*\*

\* @brief part(b) importing external class state and

\* declaring maxItems and contents of safe

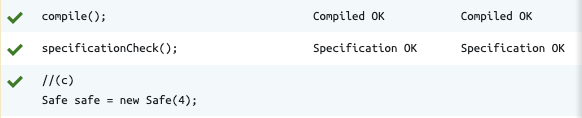
\*/

private SafeState state;

private int maxItems;

public ArrayList<String> contents;

1. CodeRunner Results



c)

1. Code

/\*\*

\* @brief part(c)(i,ii,iii)

\* @param number of maximum items used

\*/

public Safe(int aMaxItems)

{

this.state = new SafeState();

this.contents = new ArrayList<String>();

this.maxItems = aMaxItems;

}

1. CodeRunner Results



d)

1. Code

/\*\*

\* part(d) Attempt to add contents

\* @param item to add to the safe ArrayList

\*/

public void addToContents(String item)

{

if(this.contents.size() < this.maxItems)

{

contents.add(item);

}

}

1. CodeRunner Results

A white background with black text

Description automatically generated

e)

1. Code

/\*\*

\* @brief part(e) remove an item from contents

\* @param stored item for removal

\*/

public void removeFromContents(String item)

{

boolean found = false;

// index used to enable removal of item using java String remove(index)

for(int i = 0; i < this.contents.size(); i++)

{

if(contents.get(i) == item)

{

found = true;

this.contents.remove(i);

}

}

if(found)

{

System.out.println("Removed " + item);

}

else

{

System.out.println("Item " + item + " not in safe");

}

}

1. CodeRunner Results

A screenshot of a computer code

Description automatically generated

f)

1. Code

/\*\*

\* @brief part(f) Display all item contents

\*/

public void display()

{

/\*\*

\* Display all items in contents

\*/

for(String item : this.contents) // Capture and Display using for\_each

{

System.out.println(item);

}

}

1. CodeRunner Results

A white background with black text

Description automatically generated

g)

1. Code

\*\*

\* @brief part(g) Empty method to clear the array list

\*/

public void empty()

{

/\*\*

\* Carry out operation only if is not empty

\*/

if(!this.contents.isEmpty()){

/\*\*

\* Iterator to run through arraylist contents

\*/

int index = 0;

Iterator<String> contentIter = this.contents.iterator();

/\*\*

\* Iterater will run through and remove all items in contents

\*/

while(contentIter.hasNext()) {

System.out.println("Removed " + contentIter.next());

contentIter.remove();

}

}

}

1. CodeRunner Results

A screenshot of a computer

Description automatically generated

h)

1. Code

/\*\*

\* @brief part(h) Attempt to open the safe

\* @param Code to open the safe must be valid code or master code

\* @return boolean value indicating the open status of the safe

\*/

public boolean open(String aCode)

{

boolean SafeOpen = false;

if(this.state.isOpen() == true)

{

SafeOpen = true;

}

else

{

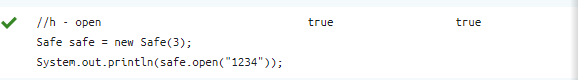
SafeOpen = this.state.open(aCode);

}

return SafeOpen;

}

1. CodeRunner Results



i)

1. Code

/\*\*

\* @brief part(i) Lock the Safe using field state object

\* @param Code passed to lock the safe(String)

\* @return boolean indicating the locked status of the safe true | false

\*/

public boolean lock(String aCode)

{

boolean isLocked = false;

if(this.state.isOpen() == false)

{

isLocked = false;

}

else

{

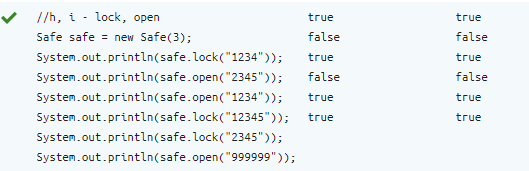
isLocked = this.state.lock(aCode);

}

return isLocked;

}

1. CodeRunner Results



# References

oracle (2019). ArrayList (Java Platform SE 8 ). [online] Oracle.com. Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>.

docs.oracle.com. (n.d.). Iterator (Java Platform SE 8 ). [online] Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html>.