Stacks Reverse Auto-Hangman Word Guess

Create an interface called Stack<anyType> with the following methods: boolean isEmpty(), void push(anyType x), anyType pop(), and anyType peek(). Then create an object called myStack that implements the interface. Then use myStack to create the following game:

Rules: Player 1 enters a word, which is then converted to a stack of single character strings. The last letter of the word is revealed. Player 2 must guess at the word one letter at a time starting from the end of the word.

Ex. The last letter is revealed, and the player guesses at the letter before it. The score is updated and the letter is revealed regardless of a correct or incorrect guess. The player then guesses at the next previous letter.

Scoring: A counter (i) is used to keep track of what letter in the word is being guessed at. If the player guesses correctly, (i) points are added to the score. If the player guesses incorrectly, then word.length() – i points are subtracted from the score. The first letter that is given at the start of the game (the last letter in the word) has no point value.

Points for guessing correctly: + 1 2 3 4 5 6

Word: h o l i d a y

Points for guessing incorrectly: - 6 5 4 3 2 1

Extra credit: Allow a 1-player version of the game where a word is chosen randomly from a file. Make a theme for each word file that may be chosen. I.e. food, places, movies, music, etc.

The theme is: South Park

Your word is: \_ \_ \_ \_ \_ \_ n

Score: 0

What letter do you guess? i

Incorrect! You lose 1 point.

Your word is: \_ \_ \_ \_ \_ a n

Score: -2

What letter do you guess? r

Incorrect! You lose 2 points.

Your word is: \_ \_ \_ \_ m a n

Score: -5

What letter do you guess? t

Correct! You get 3 points.

Your word is: \_ \_ \_ t m a n

Score: 1

**Stack HW #1 APCS Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

public interface **Stack<E>** { *//postcondition: returns true if stack is empty; otherwise, returns false*

boolean isEmpty();

*//precondition: stack is [a1, a2, ..., an] with n >= 0 //postcondition: stack is [a1, a2, ..., an, x]*

E push(E x);

*//precondition: stack is [a1, a2, ..., an] with n >= 1*

*//postcondition: stack is [a1, a2, ..., a(n-1)]; returns an.*

*//Throws an unchecked exception if the stack is empty*

E pop();

*//precondition: stack is [a1, a2, ..., an] with n >= 1 //postcondition: returns an.*

*//Throws an unchecked exception if the stack is empty.*

E peek();

}

Show the resulting stack:

Stack<String> pile1 = new Stack<String>();

pile1.push(“D”);

pile1.push(“#”);

pile1.pop();

pile1.push(“w”);

pile1.push(“H”);

pile1.pop();

Stack<String> pile2 = new Stack<String>();

pile2.push(“3”);

pile2.push(“2”);

pile2.push(“0”);

String temp = pile2.pop();

pile2.push(“7”);

pile2.push(temp);

Stack<String> pile3 = new Stack<String>();

String temp;

pile3.push(“a”);

pile3.push(“e”);

while(!pile3.isEmpty())

temp = pile3.pop();

pile3.push(temp);

pile3.push(temp);

Stack<String> pile4 = new Stack<String>();

pile4.push(“8”);

pile4.push(“2”);

pile4.push(“4”);

pile4.pop();

pile4.push(“7”);

System.out.println.(pile4.peek());

public interface **Stack<E>** { *//postcondition: returns true if stack is empty; otherwise, returns false*

boolean isEmpty();

*//precondition: stack is [a1, a2, ..., an] with n >= 0 postcondition: stack is [a1, a2, ..., an, x]*

E push(E x);

*//precondition: stack is [a1, a2, ..., an] with n >= 1 postcondition: stack is [a1, a2, ..., a(n-1)]; returns an.*

*//Throws an unchecked exception if the stack is empty*

E pop();

*//precondition: stack is [a1, a2, ..., an] with n >= 1 postcondition: returns an.*

*//Throws an unchecked exception if the stack is empty.*

E peek();

}

Consider the following stack of Integers, pile:[5, 2, 7, 9, 4, 1, 6, 3]. Consider that the object 3 is on the “top” of the stack:

Show the output of the following code:

**int** sum=0;

**while**(!pile.isEmpty())

{

**if**( pile.peek() % 2 == 0)

sum += pile.pop();

**else**

System.out.println(pile.pop() + “ “);

}

System.out.println(“The sum is:” + sum);

Write the code that will push only the odd Integers into a new stack called oddPile:

Show the resulting oddPile after running the code given the stack pile:

**Stack HW #2 APCS Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

public interface **Stack<E>** { *//postcondition: returns true if stack is empty; otherwise, returns false*

boolean isEmpty();

*//precondition: stack is [a1, a2, ..., an] with n >= 0 postcondition: stack is [a1, a2, ..., an, x]*

E push(E x);

*//precondition: stack is [a1, a2, ..., an] with n >= 1 postcondition: stack is [a1, a2, ..., a(n-1)]; returns an.*

*//Throws an unchecked exception if the stack is empty*

E pop();

*//precondition: stack is [a1, a2, ..., an] with n >= 1 postcondition: returns an.*

*//Throws an unchecked exception if the stack is empty.*

E peek();

}

Consider the following stack pile:[“5”, “$”, “7”, “@”, “4”, “\*”, “6”, “%”].

Consider that the object “%” is on the “top” of the stack:

System.out.println(pile); would show [5,$,7,@,4,\*,6,%].

1) Show the output of the following code:

Stack<String> temp = new Stack<String>();

**int** count = 1;

while(**!**pile.isEmpty())

{

**if** (count **%** 2 **==** 0)

temp.push(pile.pop());

**else**

System.out.println (“Dropped “ **+** pile.pop());

count**++;**

}

System.out.println(temp);

2) Write the code that will create a new Stack called pileRev that has all the elements of pile but the elements are in reverse order.

System.out.println(pile2); would show [%,6,\*,4,@,7,$,5].