1. Write a method in C# that takes an integer as input and prints the English words that represent that number. Example: When given 1032, the method should return “one thousand thirty two”.

Th is algorithm works by reading the numbers the same way a human would. We start by grouping them into 3-digit chunks, starting from the right. Then we determine the "words" version of that chunk (it'll be the same regardless of whether it's hundreds/thousands/millions/etc). Once we have that, we just iterate through the chunks backwards and apply the appropriate names.

public class Solution

{

private IDictionary<int, string> Numbers { get; set; } = new Dictionary<int, string>

{

{ 1, "One" },

{ 2, "Two" },

{ 3, "Three" },

{ 4, "Four" },

{ 5, "Five" },

{ 6, "Six" },

{ 7, "Seven" },

{ 8, "Eight" },

{ 9, "Nine" }

};

private IDictionary<int, string> Tens { get; set; } = new Dictionary<int, string>

{

{ 2, "Twenty" },

{ 3, "Thirty" },

{ 4, "Forty" },

{ 5, "Fifty" },

{ 6, "Sixty" },

{ 7, "Seventy" },

{ 8, "Eighty" },

{ 9, "Ninety" }

};

private IDictionary<int, string> Teens { get; set; } = new Dictionary<int, string>

{

{ 10, "Ten" },

{ 11, "Eleven" },

{ 12, "Twelve" },

{ 13, "Thirteen" },

{ 14, "Fourteen" },

{ 15, "Fifteen" },

{ 16, "Sixteen" },

{ 17, "Seventeen" },

{ 18, "Eighteen" },

{ 19, "Nineteen" }

};

private IDictionary<int, string> Names { get; set; } = new Dictionary<int, string>

{

{ 0, "" },

{ 1, " Thousand" },

{ 2, " Million" },

{ 3, " Billion" }

};

public string NumberToWords(int num)

{

if (num.Equals(0))

{

return "Zero";

}

string res = "";

IList<int> chunks = new List<int>();

while (!num.Equals(0))

{

chunks.Add(num % 1000);

num /= 1000;

}

for (int i = (chunks.Count - 1); i >= 0; i--)

{

string chunk = GetChunk(chunks[i]);

if (!string.IsNullOrEmpty(chunk))

{

res += (!string.IsNullOrEmpty(res) ? " " : "") + chunk + Names[i];

}

}

return res;

}

private string GetChunk(int num)

{

if (num > 999)

{

throw new Exception("Chunk must be 3 digits.");

}

string res = "";

int h = ((num / 100) % 10);

int t = ((num / 10) % 10);

int o = (num % 10);

if (Numbers.ContainsKey(h))

{

res += Numbers[h] + " Hundred";

}

if (t.Equals(1))

{

res += (!string.IsNullOrEmpty(res) ? " " : "") + Teens[num % 100];

}

else if (!t.Equals(0))

{

res += (!string.IsNullOrEmpty(res) ? " " : "") + Tens[t];

}

if (!t.Equals(1) && Numbers.ContainsKey(o))

{

res += (!string.IsNullOrEmpty(res) ? " " : "") + Numbers[o];

}

return res;

}

}

1. Create a class diagram for the following scenario:

*A ToasterFactory class creates Toaster instances. A Toaster has two or four slots. There are two kinds of slots: wide and regular.*

<https://www.guru99.com/uml-class-diagram.html>

<https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>

factory design patterns with multiple attributes

class ToasterFactory

{

public int slotNumber { get; set; } //two or four

public string slotType { get; set; } // wide or regular

public ToasterFactory(int slotNum, string slotT)

{

slotNumber = slotNum;

slotType = slotT;

}

}

ToasterFactory toaster = new ToasterFactory(2,"wide");

|  |
| --- |
| **ToasterFactory** |
| +slotNumber  +slotType |
|  |

1. Produce a mockup or screenshot of a one-screen web user interface for a smart toaster (that will be rendered on a 300x400 LCD screen) that meets the following user needs:
   1. Turn the toaster on/off
   2. Set the toaster’s heat level
   3. Set how long to toast

Be prepared to describe the UI behavior verbally during interview.