Kotlin

```
fun getTransactionAmounts(list: List<String>): Map<String,
Int> {
    val res = list
        .map { it.split(",") }
        .filter { it.size == 3 }
        .groupBy { it[1] }
        .mapValues { (_, values) ->
            values.sumOf { it[2].toIntOrNull() ?: 0 }
        }
        .filterValues { it >= 0 }
    return res
}

fun calculateTotalAmounts(transactions: List<String>):
Map<String, Int> {
    val resultMap = mutableMapOf<String, Int>()

    for (transaction in transactions) {
        val parts = transaction.split(",")
        if (parts.size == 3) {
            val name = parts[1]
            val amount = parts[2].toIntOrNull() ?: continue
            val currentAmount =
resultMap.getOrDefault(name, 0)
            resultMap[name] = currentAmount + amount
        }
    }

    return resultMap.filterValues { it >= 0 }
}
```

PHP:

```
function calculateTotalAmounts($transactions) {
    $resultMap = [];

    foreach ($transactions as $transaction) {
        $parts = explode(",", $transaction);
        if (count($parts) == 3) {
            $name = $parts[1];
            $amount = intval($parts[2]);
            $currentAmount = $resultMap[$name] ?? 0;
            $resultMap[$name] = $currentAmount + $amount;
        }
    }

    return array_filter($resultMap, function($value) {
        return $value >= 0;
    });
}
```

Java:

```
import java.util.*;

public class Main {
    public static Map<String, Integer>
calculateTotalAmounts(List<String> transactions) {
        Map<String, Integer> resultMap = new HashMap<>();

        for (String transaction : transactions) {
            String[] parts = transaction.split(",");
            if (parts.length == 3) {
                String name = parts[1];
                int amount = Integer.parseInt(parts[2]);
                int currentAmount =
resultMap.getOrDefault(name, 0);
                resultMap.put(name, currentAmount +
amount);
            }
        }

        return resultMap.entrySet().stream()
            .filter(entry -> entry.getValue() >= 0)
            .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));
    }
}
```

JavaScript:

```
function calculateTotalAmounts(transactions) {
    let resultMap = {};

    for (let transaction of transactions) {
        let parts = transaction.split(",");
        if (parts.length === 3) {
            let name = parts[1];
            let amount = parseInt(parts[2]);
            let currentAmount = resultMap[name] || 0;
            resultMap[name] = currentAmount + amount;
        }
    }

    return
Object.fromEntries(Object.entries(resultMap).filter(([name,
amount]) => amount >= 0));
}
```

Python:

```
def calculate_total_amounts(transactions):
    result_map = {}

    for transaction in transactions:
        parts = transaction.split(",")
        if len(parts) == 3:
            name = parts[1]
            amount = int(parts[2])
            current_amount = result_map.get(name, 0)
            result_map[name] = current_amount + amount

    return {name: amount for name, amount in
result_map.items() if amount >= 0}
```

C#:

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static Dictionary<string, int>
CalculateTotalAmounts(List<string> transactions)
    {
        Dictionary<string, int> resultMap = new
Dictionary<string, int>();

        foreach (string transaction in transactions)
        {
            string[] parts = transaction.Split(',');
            if (parts.Length == 3)
            {
                string name = parts[1];
                int amount;
                if (int.TryParse(parts[2], out amount))
                {
                    int currentAmount =
resultMap.ContainsKey(name) ? resultMap[name] : 0;
                    resultMap[name] = currentAmount +
amount;
                }
            }
        }

        return resultMap.Where(pair => pair.Value >= 0)
            .ToDictionary(pair => pair.Key, pair =>
pair.Value);
    }
}
```