## Encapsulation

**01 Pizza Calories**

A Pizza is made of a dough and different toppings.

You should model a class **Pizza** which should have a **name**, **dough** and **toppings** as fields. Every type of ingredient should have its own class.

Every ingredient has **different properties**: the dough can be **white** or **wholegrain** and in addition it can be **crispy**, **chewy** or **homemade**. The toppings can be of type **meat**, **veggies**, **cheese** or **sauce**.

Every ingredient should have a **weight** in grams and a method for calculating its calories according its type. Calories per gram are calculated through modifiers. Every ingredient has **2 calories per gram** **as a base** and a **modifier** that gives the exact calories.

**Your job** is to model the classes in such a way that they are **properly encapsulated** and to provide a public method for every pizza that **calculates its calories according to the ingredients it has**.

|  |  |
| --- | --- |
| **Dough Modifiers** | **Toppings Modifiers** |
| * White – 1.5; * Wholegrain – 1.0; * Crispy – 0.9; * Chewy – 1.1; * Homemade – 1.0; | * Meat – 1.2; * Veggies – 0.8; * Cheese – 1.1; * Sauce – 0.9; |

For example, **white** dough has a modifier of **1.5**, a **chewy** dough has a modifier of **1.1**, which means that a white chewy dough weighting **100 grams** will have (2 \* 100) \* 1.5 \* 1.1 = **330.00 total calories**.

For example, **meat** has a modifier of **1.2**, so a meat topping will have **1.2** calories per gram (1 \* 1.2).

## Data Validation

* If invalid flour type or an invalid baking technique is given an exception is thrown with the message "**Invalid type of dough.**"
* If dough weight is outside of range [1..200] throw an exception with the message "**Dough weight should be in the range [1..200]**."
* If topping is not one of the provided types throw an exception with the message "**Cannot place [name of invalid argument] on top of your pizza.**"
* If topping weight is outside of range [1..50] throw an exception with the message "[**Topping type name] weight should be in the range [1..50].**".
* If name of the pizza is empty or longer than 15 symbols throw an exception with the message "**Pizza name should be between 1 and 15 symbols.**"
* If number of topping is outside of range [0..10] throw an exception with the message "**Number of toppings should be in range [0..10].**".

The input for a pizza consists of several lines:

* On the first line is the **pizza name** and the **number of toppings it has** in format:
  + Pizza {**pizzaName**} {**numberOfToppings**}
* On the second line you will get input for the **dough** in format:
  + Dough {**flourType**} {**bakingTechnique**} {**weightInGrams**}
* On the next **N** lines, you will receive every topping the pizza has, where **N** is **number of lines for the toppings**:
  + Topping {**toppingType**} {**weightInGrams**}

If creation of the pizza was **successful** print on a single line the name of the pizza and the **total calories** it has, rounded to the second digit after the decimal point.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pizza Meatless 2  Dough Wholegrain Crispy 100  Topping Veggies 50  Topping Cheese 50  END | Meatless – 370.00 |
| Pizza Bulgarian 20  Dough Tip500 Balgarsko 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Number of toppings should be in range [0..10]. |
| Pizza Bulgarian 2  Dough Tip500 Balgarsko 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Invalid type of dough. |
| Pizza Bulgarian 2  Dough White Chewy 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Cannot place Sirene on top of your pizza. |

## 02 Football Team Generator

A football team has variable number of players, a name and a rating.

A **player** has a **name** and **stats** which are the basis for his skill level. The stats a player has are **endurance**, **sprint**, **dribble**, **passing** and **shooting**. Each stat can be in the range [0..100]. The **overall skil**l level of a player is calculated as the **average** of his stats. Only the name of a player and his stats should be visible to all of the outside world. Everything else should be hidden.

A **team** should expose a **name**, a **rating** (calculated by the average skill level of all players in the team) and **methods** for adding and removing players.

Your task is to model the team and the players following the proper principles of Encapsulation. Expose only the properties that needs to be visible and validate data appropriately.

### Data Validation

* A **name** cannot be null, empty or white space. If not, print "**A name should not be empty.**"
* **Stats** should be in the range 0..100. If not, print "**[Stat name] should be between 0 and 100.**"
* If you receive a command to **remove** a missing player, print "**Player [Player name] is not in [Team name] team.**"
* If you receive a command to **add** a player to a missing team, print "**Team [team name] does not exist.**"
* If you receive a command to **show** stats for a missing team, print "**Team [team name] does not exist.**"

|  |  |
| --- | --- |
| **Input** | **Output** |
| Team;Arsenal  Add;Arsenal;Kieran\_Gibbs;75;85;84;92;67  Add;Arsenal;Aaron\_Ramsey;95;82;82;89;68  Remove;Arsenal;Aaron\_Ramsey  Rating;Arsenal  END | Arsenal – 81 |
| Team;Arsenal  Add;Arsenal;Kieran\_Gibbs;75;85;84;92;67  Add;Arsenal;Aaron\_Ramsey;195;82;82;89;68  Remove;Arsenal;Aaron\_Ramsey  Rating;Arsenal  END | Endurance should be between 0 and 100.  Player Aaron\_Ramsey is not in Arsenal team.  Arsenal - 81 |
| Team;Arsenal  Rating;Arsenal  END | Arsenal – 0 |

**Defining Classes - Collections**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Pesho 12  Stamat 31  Ivan 48 | Ivan - 48  Stamat – 31 |
| 5  Nikolai 33  Yordan 88  Tosho 22  Lyubo 44  Stanislav 11 | Lyubo - 44  Nikolai - 33  Yordan – 88 |

**03 Optional Pull – (List – ArrayList or Set - TreeSet)**

Using the Person class - Define a class **Person** with **private** fields for **name** and **age**.

* Create constructor with parameters.
* Create **getters** and **setters.**
* Create **toString()** method

Write a program that reads from the console **N** lines of personal information and then prints all people whose **age** is **more than 30** years, **sorted in alphabetical order**.

**Note**: you can use **stream()** to filter people.

## 04 Speed Racing – (Map - LinkedHashMap)

Your task is to implement a program that keeps track of cars and their fuel and supports methods for moving the cars. Define a class **Car** that keeps track of a car’s **Model, fuel amount, fuel cost for 1 kilometer** and **distance traveled**. A Car’s Model is **unique** - there will never be 2 cars with the same model.

On the first line of the input you will receive a number **N** – the number of cars you need to track, on each of the next **N** lines you will receive information for a car in the following format “<**Model> <FuelAmount> <FuelCostFor1km>**”, all **cars start at 0 kilometers traveled**.

After the **N** lines until the command “**End**” is received (**while** (!"End".equals(command))), you will receive a commands in the following format “**Drive <CarModel> <amountOfKm>**”, implement a method in the **Car** class to calculate whether or not a car can move that distance, if it can the car’s **fuel amount** should be **reduced** by the amount of used fuel and its **distance traveled** should be increased by the amount of kilometers traveled, otherwise the car should not move (Its fuel amount and distance traveled should stay the same) and you should print on the console “**Insufficient fuel for the drive**”. After the “**End**” command is received, print each car in order of appearing in input and its current fuel amount and distance traveled in the format “**<Model> <fuelAmount> <distanceTraveled>**”, where the fuel amount should be printed to **two decimal places** after the separator.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  AudiA4 23 0.3  BMW-M2 45 0.42  Drive BMW-M2 56  Drive AudiA4 5  Drive AudiA4 13  End | AudiA4 17.60 18  BMW-M2 21.48 56 |
| 3  AudiA4 18 0.34  BMW-M2 33 0.41  Ferrari-488Spider 50 0.47  Drive Ferrari-488Spider 97  Drive Ferrari-488Spider 35  Drive AudiA4 85  Drive AudiA4 50  End | Insufficient fuel for the drive  Insufficient fuel for the drive  AudiA4 1.00 50  BMW-M2 33.00 0  Ferrari-488Spider 4.41 97 |

**05Car Salesman** (Map - HashMap)

Define two classes **Car** and **Engine.** A **Car** has a **model, engine, weight** and **color**. An Engine has **model**, **power, displacement** and **efficiency**. A Car’s **weight** and **color** and its Engine’s **displacements** and **efficiency** are **optional**.

On the first line, you will read a number **N** which will specify how many lines of engines you will receive, on each of the next **N** lines you will receive information about an **Engine** in the following format “<**Model> <Power> <Displacement> <Efficiency>**”. After the lines with engines, on the next line you will receive a number **M** – specifying the number of Cars that will follow, on each of the next **M** lines information about a **Car** will follow in the following format “<**Model> <Engine> <Weight> <Color>**”, where the engine in the format will be the **model of an existing** **Engine**. When creating the object for a **Car**, you should keep a **reference to the real engine** in it, instead of just the engine’s model, note that the optional properties **might be missing** from the formats.

Your task is to print each car (in the order you received them) and its information in the format defined bellow, if any of the optional fields has not been given print “**n/a**” in its place instead:

**<CarModel>:  
 <EngineModel>:  
 Power: <EnginePower>  
 Displacement: <EngineDisplacement>  
 Efficiency: <EngineEfficiency>  
 Weight: <CarWeight>  
 Color: <CarColor>**

### **Optional**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  V8-101 220 50  V4-33 140 28 B  3  FordFocus V4-33 1300 Silver  FordMustang V8-101  VolkswagenGolf V4-33 Orange | FordFocus:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: 1300  Color: Silver  FordMustang:  V8-101:  Power: 220  Displacement: 50  Efficiency: n/a  Weight: n/a  Color: n/a  VolkswagenGolf:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: n/a  Color: Orange |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  DSL-10 280 B  V7-55 200 35  DSL-13 305 55 A+  V7-54 190 30 D  4  FordMondeo DSL-13 Purple  VolkswagenPolo V7-54 1200 Yellow  VolkswagenPassat DSL-10 1375 Blue  FordFusion DSL-13 | FordMondeo:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: Purple  VolkswagenPolo:  V7-54:  Power: 190  Displacement: 30  Efficiency: D  Weight: 1200  Color: Yellow  VolkswagenPassat:  DSL-10:  Power: 280  Displacement: n/a  Efficiency: B  Weight: 1375  Color: Blue  FordFusion:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: n/a |

Override the classes’s ToString() methods to have a reusable way of displaying the objects.

**06 Google** (Composition - HashMap)

Google is always watching you, so it should come as no surprise that they know everything about you (even your pokemon collection), since you’re really good at writing classes Google asked you to design a Class that can hold all the information they need for people.

From the console you will receive an unkown amount of lines until the command “**End**” is read, on each of those lines there will be information about a person in one of the following formats:

* “**<Name> company <companyName> <department> <salary>**”
* “**<Name> pokemon <pokemonName> <pokemonType>”**
* “**<Name> parents <parentName> <parentBirthday>**”
* “**<Name> children <childName> <childBirthday>**”
* “**<Name> car <carModel> <carSpeed>**”

You should structure all information about a person in a class with nested subclasses. People’s names are **unique** - there won’t be 2 people with the same name, a person can also have **only 1** **company** and **car**, but can have **multiple** **parents, chidlren** and **pokemon**. After the command “**End**” is received on the next line you will receive a single name, you should print all information about that person. Note that information can change during the input, for instance if we receive multiple lines which specify a person’s company, only the **last one** should be the one remembered. The salary must be formated to **two decimal places** after the seperator.

**Note**: print the information in format:

**{personName}**

**Company:**

**{companyName} {companyDepartment} {salary}**

**...**

**Children:**

**{childName} {childBirthday}**

**{childName} {childBirthday}**

|  |  |
| --- | --- |
| **Input** | **Output** |
| PeshoPeshev company PeshInc Management 1000.00  TonchoTonchev car Trabant 30  PeshoPeshev pokemon Pikachu Electricity  PeshoPeshev parents PoshoPeshev 22/02/1920  TonchoTonchev pokemon Electrode Electricity  End  TonchoTonchev | TonchoTonchev  Company:  Car:  Trabant 30  Pokemon:  Electrode Electricity  Parents:  Children: |
| JelioJelev pokemon Onyx Rock  JelioJelev parents JeleJelev 13/03/1933  GoshoGoshev pokemon Moltres Fire  JelioJelev company JeleInc Jelior 777.77  JelioJelev children PudingJelev 01/01/2001  StamatStamatov pokemon Blastoise Water  JelioJelev car AudiA4 180  JelioJelev pokemon Charizard Fire  End  JelioJelev | JelioJelev  Company:  JeleInc Jelior 777.77  Car:  AudiA4 180  Pokemon:  Onyx Rock  Charizard Fire  Parents:  JeleJelev 13/03/1933  Children:  PudingJelev 01/01/2001 |

**07 Family Tree** (Map - HashMap)

You want to build your family tree, so you went to ask your grandmother, sadly your grandmother keeps remembering information about your predecessors in pieces, so it falls to you to group the information and build the family tree.

On the first line of the input you will receive either a name or a birthdate in the format “<**FirstName> <LastName>”** or **“day/month/year**” – your task is to find the person’s information in the family tree. On the next lines until the command “**End**” is received you will receive information about your predecessors that you will use to build the family tree.

The information will be in one of the following formats:

* “**FirstName LastName - FirstName LastName**”
* “**FirstName LastName - day/month/year**”
* “**day/month/year - FirstName LastName**”
* “**day/month/year - day/month/year**”
* “**FirstName LastName day/month/year**”

The first 4 formats reveal a family tie – **the person on the left is parent to the person on the right** (as you can see the format does not need to contain names, for example the 4th format means the person born on the left date is parent to the person born on the right date). The last format ties different information together – i.e. **the person with that name was born on that date**. **Names** and **birthdates** are **unique** – there won’t be 2 people with the same name or birthdate, there will **always** be enough entries to construct the family tree (all people’s names and birthdates are known and they have atleast one connection to another person in the tree).

After the command “**End**” is received you should print all information about the person whose name or birthdate you received on the first line – his **name, birthday, parents and children** (check the examples for the format). The people in the parents and childrens lists should be ordered by their first appearance in the input (regardless if they appeared as a birthdate or a name, for example in the first input Stamat is before Penka because he first appeared in the second line, while she appears in the third.).

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Peshev  11/11/1951 - 23/05/1980  Penka Pesheva - 23/05/1980  Penka Pesheva 09/02/1953  Pesho Peshev - Gancho Peshev  Gancho Peshev 01/01/2005  Stamat Peshev 11/11/1951  Pesho Peshev 23/05/1980  End | Pesho Peshev 23/05/1980  Parents:  Stamat Peshev 11/11/1951  Penka Pesheva 09/02/1953  Children:  Gancho Peshev 01/01/2005 |
| 13/12/1993  25/03/1934 - 04/04/1961  Poncho Tonchev 25/03/1934  04/04/1961 - Moncho Tonchev  Toncho Tonchev - Lomcho Tonchev  Moncho Tonchev 13/12/1993  Lomcho Tonchev 07/07/1995  Toncho Tonchev 04/04/1961  End | Moncho Tonchev 13/12/1993  Parents:  Toncho Tonchev 04/04/1961  Children: |

**08 Cat Lady (**Map - HashMap**)**

Ginka has many cats in her house of various breeds, since some breeds have specific characteristics, Ginka needs some way to catalogue the cats, help her by creating a class hierarchy with all her breeds of cats so that she can easily check on their characteristics. Ginka has 3 specific breeds of cats “Siamese”, “Cymric” and the very famous bulgarian breed “Street Extraordinaire”, each breed has a specific characteristic about which information should be kept. For the Siamese cats their ear size should be kept, for Cymric cats - the length of their fur in milimeters and for the Street Extraordinaire the decibels of their meowing during the night.

From the console you will receive lines of cat information until the command “**End**” is received, the information will come in one of the following formats:

* “**Siamese <name> <earSize>”**
* “**Cymric <name> <furLength>”**
* “**StreetExtraordinaire <name> <decibelsOfMeows>”**

On the last line after the “**End**” command you will receive the name of a cat, you should print that cat. Round the numbers **two digits** after the decimal separator.

**Hint:**

Use class inheritance to represent the cat hierarchy and override the ToString() methods of concrete breeds to allow for easy printing of the cat, regardless what breed it is.

|  |  |
| --- | --- |
| **Input** | **Output** |
| StreetExtraordinaire Maca 85  Siamese Sim 4  Cymric Tom 28  End  Tom | Cymric Top 28.00 |
| StreetExtraordinaire Koti 80  StreetExtraordinaire Maca 100  Cymric Tim 31  End  Maca | StreetExtraordinaire Maca 100.00 |