Aplicatie - Imparte surprize

Deoarece niciodata nu este gresit sa daruiesti, in acest capitol vom implementa o aplicatie menita sa gestioneze un set de surprize.

Scop final

Scopul final este de a obtine un **sistem generic** capabil sa:

1. utilizeze mai multe tipuri de surprize (i.e. ce implementeaza o interfata comuna)
2. implementeze diversi algoritmi de stocare si daruire a surprizelor
3. foloseasca un mecanism puternic, aleator de generare a unui set (nou) de surprize

Genericitate

Orice programator software de elita implementeaza aplicatii uitandu-se mereu la viitor. Deoarece nimeni nu poate cunoaste cu precizie cum va evolua mediul real si implicit programul scris, aplicatia va fi dezvoltata pornind de la niste **specificatii abstracte de design**, pentru fiecare tip de componenta in parte.

Daca pana acum am dezvoltat aplicatii avand in vedere functionalitatea finala (i.e. implementarea efectiva a fiecarei clase, metode etc.) target-ul actual este mult mai inalt. Concret, aplicatia curenta va fi dezvoltata pornind de la premisa de **genericitate** si capacitatea de a defini noi componente sau de a actualiza pe cele existente **cu usurinta**, **intr-un mod transparent** fata de restul aplicatiei.

Mai exact, considerand [scopul final](https://edu.devmind.ro/java-2/java-class-hierarchy-exercise#scop_final) al aplicatiei dorim sa putem:

1. defini cu usurinta noi tipuri de dorinte, care pot fi usor integrate cu cele deja existente
2. crea noi algoritmi de gestiune a unei colectii de surprize, care ulterior sa poata inlocui sau extinde setul existent fara modificari majore in codul existent
3. lucra in mod comun cu orice tipuri de surprize
4. gestiona surprizele intr-un mod generic, indiferent de tipul (actual) de container

Arhitectura

1. Tipuri de surprize

Fiecare surpriza va implementa urmatoarea interfata, ce permite „deschiderea“ surprizei:

[**ISurprise**](https://edu.devmind.ro/_export/code/java-2/java-class-hierarchy-exercise?codeblock=0)

package surprise;

public interface ISurprise {

*// Opens the surprise and enjoys it*

void enjoy();

}

Tipurile de surprize sunt urmatoarele:

| **Surpriza** | **Detalii** | |
| --- | --- | --- |
| FortuneCookie | Reprezinta o surpriza de tipul *Fortune Cookie* in care persoana va primi un mesaj simbolic, care sa o calauzeasca in viitor. | |
| **Operare**: | Clasa primeste mesajul (i.e. String) la constructia obiectului. Acesta va fi afisat la output in momentul „deschiderii surprizei“. |
| Candies | Surpriza contine un numar **aleator** de bomboane, de un anumit tip. | |
| **Operare**: | Constructorul clasei va primi numarul de bomboane si tipul acestora. La executia surprizei se va afisa un mesaj relevant, in functie de starea interna a obiectului-surpriza. |
| MinionToy | Surpriza va fi o jucarie Minion. | |
| **Operare**: | Clasa primeste la crearea obiectelor numele minionului. La deschiderea surprizei, se va afisa un mesaj sugestiv la output, ce include detaliile jucariei primite. |

Fiecare clasa-surpriza va contine si o metoda statica generate(), care va crea o surpriza de tipul respectiv. Pentru fiecare tip de surpriza, detaliile algoritmului de generare sunt:

| **Surpriza** | **Algoritm de generare** |
| --- | --- |
| FortuneCookie | Clasa va contine un vector de minim 20 de zicale (+ autorul acestora) la alegere. Poti lua colectia de zicale de pe un site de specialitate. La apelul metodei generate(), metoda va [genera un numar aleator](https://edu.devmind.ro/java-2/java-class-hierarchy-exercise#generarea_numerelor_aleatoare_in_java) si va alege elementul din array in consecinta. Ulterior, va crea o (noua) surpriza corespunzator. |
| Candies | Clasa va mentine intern o colectie de tipuri de bomboane. De exemplu: [chocolate, jelly, fruits, vanilla]. La generarea unei noi surprize, metoda va [genera 2 numere aleatoare](https://edu.devmind.ro/java-2/java-class-hierarchy-exercise#generarea_numerelor_aleatoare_in_java), pentru a decide tipul bomboanelor (i.e. o surpriza contine un singur tip de bomboane) si respectiv numarul acestora. Ulterior, va crea o (noua) surpriza corespunzator. |
| MinionToy | Clasa va stoca o colectie de nume de minioni. De exemplu: [Dave, Carl, Kevin, Stuart, Jerry, Tim]. La generarea unei noi surprize, se va alege **in ordine** urmatorul tip de minion, conform colectiei ordonate prezentate anterior. Concret, prima surpriza generata va oferi un minion Dave, urmatoarea un Carl, s.a.m.d.. → **Hint**: vei avea nevoie de un contor intern pentru a retine ultimul minion oferit. |

Generarea numerelor aleatoare in Java

In Java, se pot genera numere aleatoare folosind biblioteca [Java API - Random](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Random.html). Analizand metodele disponibile, observam ca putem genera numere reale, intregi, valori logice, s.a.m.d.

Un exemplu de utilizare pentru generarea de numere intregi este:

import java.util.Random;

public class Main {

public static void main([String](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {

[Random](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+random) random = new [Random](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+random)();

int n = random.nextInt();

[System](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(n);

[System](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(random.nextInt());

[System](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(random.nextInt());

}

}

[Exemplu de Output](https://edu.devmind.ro/java-2/java-class-hierarchy-exercise#folded_9a8dd3be1e5d4b1cd4e3d00847d00422_1)

[**Console**](https://edu.devmind.ro/_export/code/java-2/java-class-hierarchy-exercise?codeblock=2)

-1871838097

690475167

-1794910375

Intr-un mod similar se folosesc si alte metode din API-ul clasei. Alege metoda potrivita pentru scenariul actual.

O implementare ineficienta ar crea un obiect de tipul Random la fiecare apel al metodei generate(). Tinand cont ca acelasi obiect random poate fi reutilizat, gandeste-te la o implementare eficienta.

2. Depozitarea surprizelor

Toate surprizele vor fi depozitate in diverse tipuri de containere, care vor dicta modul de stocare si mai ales **ordinea in care surprizele vor fi impartite**. Deoarece dorim sa operam cu orice tip de container intr-un mod **transparent**, independent de implementare, vom utiliza **o interfata** ce va defini protocolul concret de comunicare. Astfel:

1. **Implementare**: fiecare container va implementa interfata comuna
2. **Utilizare**: fiecare entitate-utilizator va folosi un container din perspectiva interfetei, utilizand implicit numai metodele definite de acea interfata.

Interfata va defini urmatoarele metode:

[**IBag.java**](https://edu.devmind.ro/_export/code/java-2/java-class-hierarchy-exercise?codeblock=3)

public interface IBag {

*// adds a surprise in the bag*

void put(ISurprise newSurprise);

*// adds all the surprises from another IBag*

*// -> the 'bagOfSurprises' will be empty() afterwards*

void put(IBag bagOfSurprises);

*// removes a surprise from the bag and returns it*

ISurprise takeOut();

*// Checks if the bag is empty or not*

boolean isEmpty();

*// Returns the number of surprises*

int size();

}

Se implementeaza urmatoarele tipuri de „tolbe de surprize“:

1. O tolba care va da cadourile din interior intr-o ordine aleatoare
2. O tolba care va da cadourile in ordinea in care acestea au fost introduse (strategie numita [FIFO](https://en.wikipedia.org/wiki/FIFO_and_LIFO_accounting#FIFO))
3. O tolba care va da cadourile in ordinea inversa introducerii (strategie numita [LIFO](https://en.wikipedia.org/wiki/FIFO_and_LIFO_accounting#LIFO))

Pentru implementarea diversilor containeri, poti folosi o strategie similara ca in proiectul [Sistem de gestiune înscrieri](https://edu.devmind.ro/java-2/proiect-sistem-gestiune-inscrieri), unde stocarea elementelor se realiza intr-un [ArrayList](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/ArrayList.html).

3. Crearea surprizelor

Pentru a genera un set de surprize **aleatoare**, vom implementa o clasa „speciala“, GatherSurprises, care va contine:

1. o metoda **statica**, gather(int n), ce primeste un numar intreg n si returneaza un array continand n surprize.
2. o metoda **statica**, gather(), care va returna o singura surpriza.

Observatii:

* desi intern metodele genereaza aleator surprizele, aceasta simbolizeaza actiunea reala de a colecta surprizele pregatite de oameni darnici pentru cei mici (i.e. in preajma sarbatorilor).
* Deoarece fiecare om pregateste ce surpriza doreste, colectia / surpriza intoarsa va fi de fiecare data diferita.
* Se utilizeaza 2 mecanisme de generare aleatoare:
  1. cel implementat deja, prin care se genereaza aleator o surpriza apartinand unui anumit tip
  2. un numar **random** pentru a se alege tipul de surpriza generata, la fiecare pas

Specificatii

Prin definitie, clasa GatherSurprise nu ar trebui sa permita crearea instantelor si nici mostenirea, deoarece implementarea sa este definitiva si toate metodele sale sunt statice. Desi Java nu permite marcarea unei clase static, putem sa recurge la urmatoarele artificii:

1. **instantiere**: putem crea un (singur) constructor, de forma, pe care sa il marcam private.
   * Obs: Intr-un astfel de caz, trebuie sa avem grija ca toate entitatile clasei sa fie **statice**, altfel acestea nu vor putea fi accesate vreodata.
2. **mostenire**: ce keyword folosim pentru a impiedica extiderea clasei?

4. Crearea containerelor

Deoarece exista mai multe tipuri de containere, ne dorim implementarea unui mecanism unic prin care sa generam un nou container (gol). Un astfel de sablon de programare (eng: ***design pattern***) este unul des utilizat in programare si se numeste [Factory Pattern](https://en.wikipedia.org/wiki/Factory_method_pattern).

Interfata comuna pentru orice **builder** de containere va fi (in cazul de fata):

[**IBagFactory.java**](https://edu.devmind.ro/_export/code/java-2/java-class-hierarchy-exercise?codeblock=4)

public interface IBagFactory {

*// Create a new container, according to the specified type*

IBag makeBag([String](https://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) type);

}

In functie de valoarea String-ului, se va crea o noua colectie. Tipurile posibile corespund contrainerelor definite in [sectiunea anterioara](https://edu.devmind.ro/java-2/java-class-hierarchy-exercise#depozitarea_surprizelor):

* "RANDOM"
* "FIFO"
* "LIFO"

Existenta unei interfete pentru crearea de obiecte-container permite definirea mai multor „fabrici“ de tipuri de bag-uri, care pot fi optimizate pentru diverse constrangeri (i.e. viteza de acces, spatiul de stocare etc.). De exemplu, putem defini 2 tipuri de fabrici de obiecte:

* class BagFactoryOptimizeStorage implements IBagFactory: care va construi containere optimizate pentru stocare
* class BagFactoryOptimizeAccess implements IBagFactory: care va construi containere optimizate pentru viteza de accesare a elementelor

Implementare

In cadrul aplicatiei, crearea bag-ului se va face prin intermediul unei instante a clasei care implementeaza IBagFactory. Este suficient sa avem un singur tip de *Factory* deoarece avem o singura categorie de stocare (i.e. cea bazata pe ArrayList).

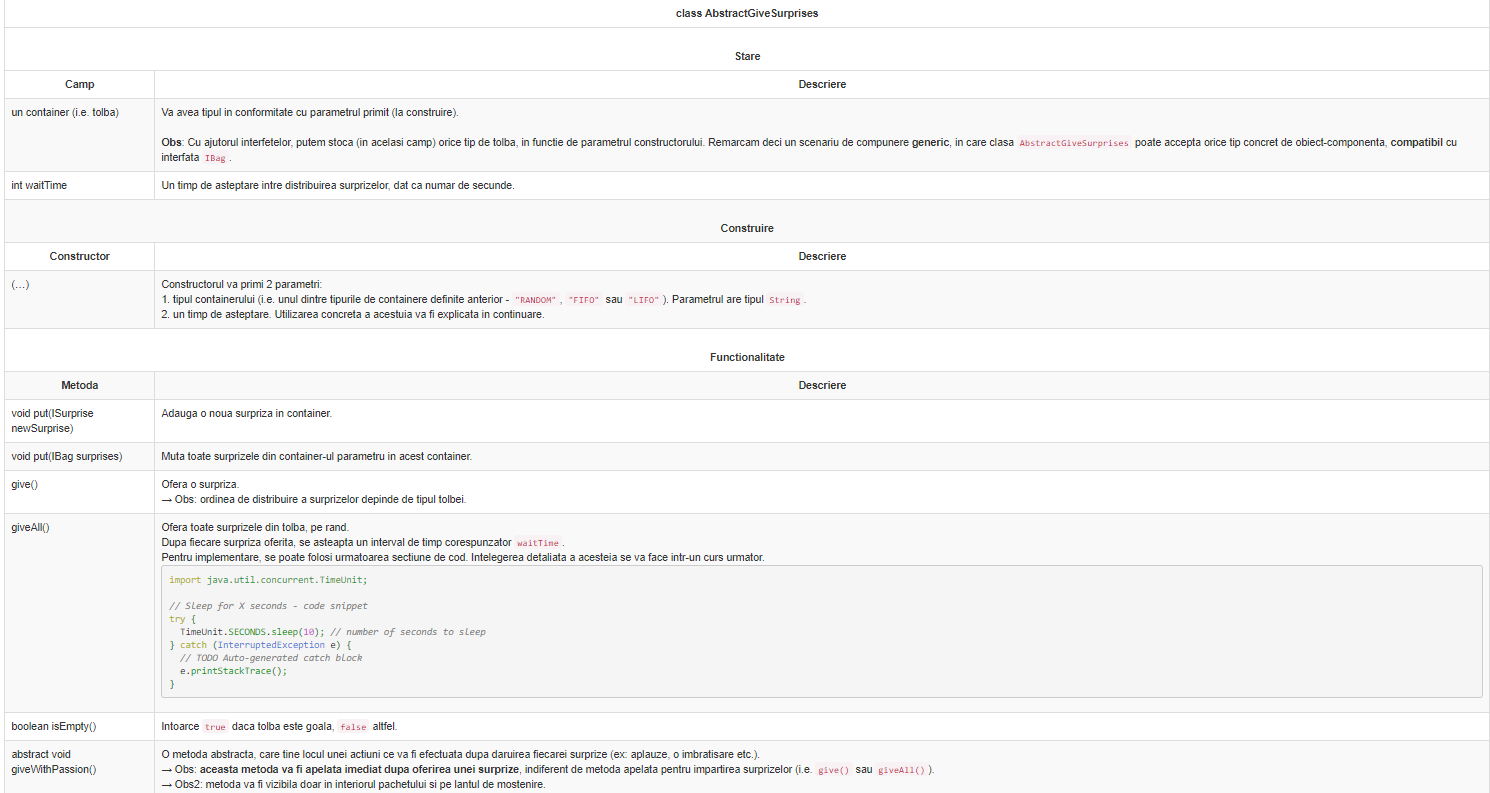
* Obs: incearca sa eviti instantierea multipla a clasei definite. Foloseste aceeasi instanta pentru generarea oricator containere.

5. Impartirea surprizelor

Pana in acest pas, am implementat: (1) diverse tipuri de surprize, (2) cateva tipuri de containere pentru stocarea acestora si (3) un generator aleator, automat, de surprize. Insa ce rost au toate acestea fara **bucuria de a darui**?

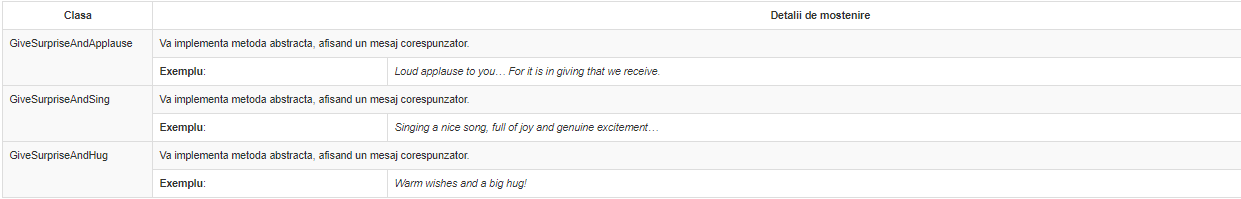
5.1. Impartirea surprizelor - mecanism abstract

In cadrul acestui task, vom implementa un mecanism de a imparti surprizele, **utilizandu-ne de functionalitatea implementata anterior**. Pentru aceasta, se defineste clasa abstracta AbstractGiveSurprises, conform urmatoarelor specificatii:



5.2. Impartirea surprizelor - moduri posibile

Clasa abstracta va fi extinsa de 3 tipuri de clase normale, dupa cum urmeaza:



La final...

Aminteste-ti fiecare **componenta abstracta** implementata in cadrul sistemului:

1. interface ISurprise
2. interface IBag
3. interface IBagFactory
4. abstract class AbstractGiveSurprises

In continuare, gandeste-te la rolul fiecarei entitati abstracte si incearca sa raspunzi la urmatoarele intrebari:

1. Ce rol are componenta in sistemul existent?
   1. La ce a contribuit in ecosistemul aplicatiei?
2. Ce tipuri de entitati depind direct de entitatea abstracta?
3. Ce rol joaca interfata / clasa abstracta in cazul in care se doreste extinderea sistemului?
   1. Ce (noi) entitati vor trebui sa respecte sau sa integreze specificatiile componentei?
   2. Care entitati beneficiaza de pe urma existentei componentei, asigurandu-se faptul ca acestea nu trebuie modificate odata cu definirea unui nou tip de element (cnf. subpunctului anterior). Cu alte cuvinte, caror entitati li se asigura (i.e. prin entitatea abstracta) transparenta utilizarii?