Упражнения : Комуникация между обекти и събития

Problem 1. Реализация на събитие

Създайте клас **Dispatcher** със свойство **name** и клас **Handler**. Създайте публичен делегат , наречен **NameChangeEventHandler**, връщащ тип void в именното пространство (namespace) na Dispatcher (Но вън от класа Dispatcher) и **събитие** (поле от типа на делегата) вътре в класа **Dispatcher** с име **NameChange**. Създайте клас NameChangeEventArgs, който наследява класа EventArgs и има свойство – **name,** което се получава през конструктора и има private сетър (setter) и public гетър (getter). Създайте също и метод, наречен **OnNameChange(NameChangeEventArgs args)** в **Dispatcher** – това е метод, който ще бъде изикван да „запали“ събитието. В сетъра за име на Dispatchera, извикайте метода **OnNameChange** и го предайте на обект от тип NameChangeEventArgs с новата стойност на имет, предадено в сетъра.

Напишете метод **OnDispatcherNameChange(object sender, NameChangeEventArgs args)** в класа Handler, реаизацията трябва да извежда на конзолата **“Dispatcher’s name changed to <newName>”**. При стартиране на програмата, създайте нов Dispatcher и Handler, тогава добавете метод **OnDispatcherNameChange** на бработчика (Handler) към **събитието NameChange в класа** Dispatcher.

Вход

От конзолата ще получите редове, съдържащи имена, докато получите командата "Край" . За всяка промяна на име, се сменя името на диспечера с това. Винаги кгато името на диспечера се променя, вие трябва да запалите събитие за всички наблюдатели (слушатели, абонати).

Изход

при всяка промяна на име на диспечера, обработчика да извежда на екарана **“Dispatcher's name changed to <newName>.”**

Ограничения

- Имената да съдържат само букви.
- Броят команди да е цяло число в интервала **[1…100]**.
- Последната команда винаги да е команда **“End”**.

Примери

Вход	Изход
Pesho Gosho Stefan End	Dispatcher's name changed to Pesho. Dispatcher's name changed to Gosho. Dispatcher's name changed to Stefan.
Prakash Stamat MuadDib Ivan Joro End	Dispatcher's name changed to Prakash. Dispatcher's name changed to Stamat. Dispatcher's name changed to MuadDib. Dispatcher's name changed to Ivan. Dispatcher's name changed to Joro.

Problem 2. Царски гамбит

Направете 3 класа - **King, Footman** и **Royal Guard**. Всички имат **name** (имената са **уникални.** Няма две единици с едно и също име), Слугите и царските охранители може също да бъдат убивани (убитите се премахват от програмата), **докато е възможно** да се атакува царя –трябва да имате метод за отговор на атаките. Когато царяте атакуван, той трябва да ивежда н конзолата "**King <kingName> is under attack!**" и всички **живи** слуги и царски пазачи трябва да отгворят на атаката:

- **Слугата** отговаря, като извежда на екрана **"Footman <footmanName> is panicking!"**.
- **Царският охранител** извежда Вместо това **"Royal Guard <guardName> is defending!"**.

Вход

В първия ред на конзолата ще получите един низ - името на **царя**. На втория ред ще получите имената на неговите царски охранители, разделени с интервали. На третия имената на неговите слуги, разделени с интервали. На следващите редове, докато се получи командата "Край", вие ще получите команди в следния формат:

- **"Attack King"** – извиква отговор на царя към атакуващия.
- **"Kill <name>"** – слуга или охранител с даденото име да се убие

Изход

Когато царя се атакуватрябва да изведете на конзолата "**King <kingName> is under attack!**" и всеки **жив** слуга и охранител (пазач) извежда **тяхното съобщение за отговор** – първо всички пазачи трябва да отговорят (в реда, който с въведени) и тогава слугите в същия ред. Всяко съобщение се извежда на нов ред.

Ограничения

- Имената ще съдържат само букви.
- Да им винаги **един цар и поне един слуга и един охранител**.
- Царят **не се убива** – няма команда за убиване на царя.
- Командите за убиване се получават само за живи войни
- Всички команди се получават в посочения формат
- Броят на командите ще е положително цяло число между **[1…100]**.
- Последната команда винаги е **"End"**.

Примери

Вход	Изход
Pesho Krivogled Ruboglav Gosho Pencho Stamat Attack King End	King Pesho is under attack! Royal Guard Krivogled is defending! Royal Guard Ruboglav is defending! Footman Gosho is panicking! Footman Pencho is panicking! Footman Stamat is panicking!
HenryVIII Thomas Oliver Mark Kill Oliver Attack King Kill Thomas	King HenryVIII is under attack! Royal Guard Thomas is defending! Footman Mark is panicking! King HenryVIII is under attack!

Kill Mark Attack King End	

Problem 3. Избягване на зависимостите

Даден ви е скелет на прост проект. Проектът съдържа клас примитивен калкулатор, който поддържа два метода - **ChangeStrategy (char @operator) и PerformCalculation (int firstOperand, int secondOperand)**. Методът **PerformCalculation** трябва да извършва математически операции върху два операнда, въз основа на текущата стратегия на примитивния калкулатор и **ChangeStrategy** трябва да промени текущата стратегия на калкулатора. В момента Калкулаторът поддържа само добавяне и изваждане на стратегии, помислете как да промените (рефакторирате) ChangeStrategy и PerformCalculation метод за да позволи на примитивния калкулатор да поддържа всякакви стратегии. Добавете функционалност към примитивния калкулатор, за да поддържа умножение и деление на елементи.

Калкулаторът трябва да стартира по **подразбиране** в режим **addition** (събиране). Текущият метод **ChangeStrategy** превключва само между 2 стратегии, основани на получаване на символ чрез метод. В момента поддържаните стратегии са:

- **"+"** за събиране
- **"-"** за изваждане

Вход

От клавиатурата вие ще получите редове в един от следните формати, до получаване ан кманда **"End"**:

- **"<number> <number>"** – изпълнява изчиление на текущите числа с текущия режим изчисление.
- **"mode <operator>"** – променя режима на изчисляване към указания.

Изход

Извежда резултата от изчислението На всички редове с чила – всеки резултат на нов ред.

Ограничения

- Позволено Ви е да промените (рефактрирате) класа Primitive Calculator, но НЕ Ви е позволено да добавяте допълнителни методи към него като например метод Addition, Subtraction и т.н.
- **Операторите,** получени от конзолата винаги ще бъдат валидни, указани в секцията за спецификации.
- **Резултатът от изчислението** също ще бъде **цяло число.**
- Делител О **не може** да има.
- Винаги се завършва с команда **"End"**.

Примери

Вход	Изход
10 15 mode / 20 5 17 7 mode - 30 31 End	25 4 2 -1

mode * 1 1 3 21 -5 -6 mode - -30 -50 mode / -28 4 mode + 1 10 End	1 63 30 20 -7 11

Problem 4. * Работна сила

Създайте два класа - **StandartEmployee** и **PartTimeEmployee** и двата да имат **име** и **работни часове седмично**. За **StandartEmployee** работните часове на седмица са винаги **40**, а за **PartTimeEmployee** работните часове на седмица са винаги **20**. Създайте клас **Job**, който трябва да получи служител чрез конструктора си, има полета - **name** и **hours of work required** и метод **Update**, който трябва да изважда от часовете на работа на служителя работните часове на седмица. Когато **hours of work required** достигне 0 или по-малко трябва да отпечатате **"Job <jobName> done!"** и да намерите начин да уведомите колекцията, в която държите всички работи, че е готова и следва да бъдат заличени от колекцията.

Вход

От конзолата ще получите редове в един от следните формати докато получите команда "Край":

- **"Job <nameOfJob> <hoursOfWorkRequired> <employeeName>"**- ще създаде работа (Job) със указаното име, задължителни седмични и служител.
- **"StandartEmployee <name>"** – ще създаде стандартен служител с указаното име.
- **"PartTimeEmployee <name>"** – ще създаде служител с парциално работно време (**PartTimeEmployee**) с указаното име.
- **"Pass Week"** – ще извика метода **Update** на всяка работа(дейност,job).
- **"Status"** – ще изведе статуса на всички работив седния формат **"Job: <jobName> Hours Remaining: <hoursOfWorkRequired>"**.

Изход

Винаги всяка работа завършва със съобщението **"Job <jobName> done!", което ще се изведе на конзолата. Винаги командата Status** показва всички **текущо активни работи** (незавършени) те ще бъдат изведени в формата, указан за **Status**, в реда на получаване – всяко съобщение на отделен ред.

Ограничения

- Всички имена съдържат само букви
- Всички задължителни часове на седмица трябва да са положителни цели числа между **[1…1000]**.
- Служителите, указани на входния ред за работа (дейност) винаги трябва да са валидни съществуващи служители.
- Служителите и названията на работите са уникални – няма служители или работи(дейности) с еднакви имена.
- Последната команда винаги е **"End"**.

Примери

Вход	Изход
StandartEmployee Pesho PartTimeEmployee Penka Job FeedTheFishes 45 Pesho Pass Week Status Pass Week End	Job: FeedTheFishes Hours Remaining: 5 Job FeedTheFishes done!
PartTimeEmployee Penka PartTimeEmployee Vanka PartTimeEmployee Stanka Job Something 177 Stanka Pass Week Job AnotherThing 33 Vanka Status Pass Week Pass Week Pass Week Status End	Job: Something Hours Remaining: 157 Job: AnotherThing Hours Remaining: 33 Job AnotherThing done! Job: Something Hours Remaining: 97

Подсказка

Намерете начин да имате своя колекция, която да реагира на събития. Създайте свой собствен клас разширяващ ArrayList и реализиращ **EventListener** за потребителски събития, който се задейства, когато работата е свършена. Използвайте абстракция в класа job за да позволите да бъдат приемани различни типове – например извлечете интерфейс за служители и ще имате клас, приемащ като обект от тях, реализиращ интерфейса вместо конкретен клас

Problem 5. *Царски гамбит (разширен)

Разширете кода си от проблем 2 King's гамбит - нормално слугите, които сега трябва да умрат в 2 удара (трябва да получи 2 команди Kill с име от входа, които да се убият), докато охранителите рябва да умират от 3 удара. Мъртвите слуги и охранители вече няма да отговарят на събития и трябва да се изтрият от колекцията. Намерете начин за умиращите войниците да съобщят за тяхната смърт на царя и колекцията , която ги съдържа, без ръчно проверка на тяхното състояние на всяка Kill команда (т.е. използвайте събития).

Вход

В първия ред на конзолата ще получите един низ - името на краля. На втория ред ще получите имената на неговите пазачи, разделени с интервали. На третия имената на неговите слуги, разделени с интервали. На следващите редове, докато получите команда "Край" , ще получавате команди в един от от следните формати:

- **“Attack King”** – извиква метод за отговор на атаката срещу царя.
- **“Kill <name>”** – слугата или охранителя с даденото име, който е атакуван, ако е втора за слуга или треа за охранител – бива убиван

Изход

Когато царят е атакуван трябва да печатате на конзолата “**King <kingName> is under attack!**” и всеки **жив** слуга и царски охранител трябва да изведе **своето ответно съобщение** - първо, всички кралски охрантели

трябва да отговорят (в ред,а в който са въведени) и след това всички слуги трябва да отговорят (в реда, в който са въведени). Всяко съобщение, трябва да бъде отпечатано на нов ред.

Ограничения

- Имената ще съдържат само букви.
- Да има винаги **един цар и поне един слуга и един охранител**.
- Царят **не се убива** – няма команда за убиване на царя.
- Командите за убиване се получават само за живи войни
- Всички команди се получават в посочения формат
- Броят на командите ще е положително цяло число между **[1…100]**.
- Последната команда винаги е **“End”**.

Примери

Вход	Изход
Pesho Ruboglav Gosho Stamat Kill Gosho Kill Stamat Attack King Kill Gosho Attack King End	King Pesho is under attack! Royal Guard Ruboglav is defending! Footman Gosho is panicking! Footman Stamat is panicking! King Pesho is under attack! Royal Guard Ruboglav is defending! Footman Stamat is panicking!
HenryVIII Thomas Mark Kill Thomas Kill Mark Attack King Kill Thomas Kill Thomas Kill Mark Attack King End	King HenryVIII is under attack! Royal Guard Thomas is defending! Footman Mark is panicking! King HenryVIII is under attack!


SoftUni
Foundation

BY NC SA