Si risolva l'esercizio proposto. Per facilitare la correzione, se possibile includere tutto in un unico file sorgente. La sufficienza è raggiunta risolvendo correttamente i primi quattro punti.

Esercizio

Si vuole abbozzare un piccolo framework per descrivere i rimbalzi in un flipper (tra palline e *bumpers*).

1. Si scriva una classe `Ball` che rappresenterà una palla. Tra i membri *private* si mettano le due componenti del vettore velocità e la massa (si usino variabili `double`: per semplicità trascureremo le unità di misura). Tra i membri *public* si scriva un costruttore che richieda come parametri massa e componenti della velocità, con valori di default tutti e tre uguali a 1.

2. Si scrivano due funzioni (membri di `Ball`): una funzione `energy` che restituisca il valore dell'energia cinetica, e una funzione `print` che stampi su `cout` le componenti della velocità e l'energia cinetica (usando `energy`).

3. Si scriva una funzione `change_speed` (anch'essa membro) che prenda come parametro un numero reale λ e riscali di un fattore λ il modulo della velocità, mantenendone invariata la direzione.

4. Nel `main` si istanzi un oggetto di tipo `Ball` di massa 1 e velocità $(1, 1)$, se ne stampino componenti ed energia cinetica, e si verifichi che se si raddoppia la velocità con `change_speed` l'energia quadruplica.

5. Si scriva poi una classe `Bumper` che rappresenterà i respingenti, cioè bersagli su cui la palla può rimbalzare. Nell'urto, la palla inverte il verso del suo moto e aumenta la sua energia cinetica di un fattore μ (cioè $E \mapsto \mu E$). Si ponga μ come membro *private* e si scriva un opportuno costruttore che prenda un parametro e inizializzi μ.

6. Tra i membri *public* di `Bumper` si scriva una funzione `bounce`, che prenda come parametro una palla e le faccia compiere l'urto descritto al punto precedente. Tale funzione dovrà essere dichiarata *virtuale* per risolvere i punti successivi. [Si valuti se passare la palla per copia o per referenza.]

7. Si scriva una classe `ThresholdBumper`, che erediti pubblicamente da `Bumper`. L'urto con questo particolare respingente ha il comportamento usuale (con $\mu = 1.5$ fissato) se l'energia cinetica supera una soglia ϵ, altrimenti è un urto elastico (conserva l'energia). Si ponga ϵ tra i membri *private* e si scriva un opportuno costruttore che prenda come parametro la soglia e inizializzi sia ϵ che il parametro μ della classe base.

8. Si scriva la funzione `bounce` (*override* di quella ereditata da `Bumper`) in modo che esegua l'urto con il controllo sulla soglia descritto al punto precedente. (Si riutilizzi il codice già scritto per la funzione della classe base.)

9. Si scriva una funzione globale `flipper` che prenda due parametri: una referenza a `Ball` e un `std::vector` di respingenti. Sapendo che l'obiettivo è quello di realizzare un comportamento polimorfico, si consideri se usare un vettore di referenze, di puntatori, oppure di copie. L'effetto di `flipper` deve essere quello di far eseguire alla palla i rimbalzi con tutti i respingenti nel vettore (in ordine).

10. Nel `main`, istanziare due `std::vector` (di referenze, puntatori o copie, vedi punto precedente). Riempire il primo con oggetti di tipo `Bumper` allocati *dinamicamente* con parametri μ lanciati a caso tra 1 e 2. Riempire il secondo con oggetti di tipo `ThresholdBumper` (anch'essi dinamici), con parametri ϵ lanciati a caso tra 0 e 5. Per ognuno dei vettori, eseguire il flipper a partire da una palla inizializzata coi valori di default, poi stampare lo stato finale della palla.

    [Per generare numeri reali pseudo-random tra 0 e 1 si può usare `drand48()`.]

    [Ricordarsi di chiamare `delete` su ogni oggetto creato con `new`.]

11. Verificare che lo stato finale della palla dopo il flipper con i `Bumper` non cambia se si inverte l'ordinamento dei respingenti, mentre lo fa con i `ThresholdBumper`.