Individual Assignment 2 – Wonderland

In a board game Wonderland several players take turns to achieve the main goal – to escape the Wonderland. The player who escapes the Wonderland first wins.

Game board and tiles

The game board is composed of a set of interconnected tiles:

```
Board b;
auto t1 = std::make_shared<Tile>("Narrow path");
auto t2 = std::make_shared<Tile>("Old tree");
auto t3 = std::make_shared<Tile>("Rabbit hole");
auto t4 = std::make_shared<Tile>("Bottom of rabbit hole");
auto t5 = std::make_shared<Tile>("Wonderland woods");
auto exit = std::make_shared<ExitWonderlandTile>();
b.append(t1).append(t2).append(t3).append(t4).append(t5).append(exit);
```

There is a special tile **ExitWonderlandTile**, where users can enter if they have specific types of playing cards. For class **Tile**, implement method **canEnter(Player)** which returns **true** if a given player can enter a given tile, **false** otherwise.

Players, player moves, move behaviors

Players must be aware of their location:

```
std::unique_ptr<Player> alice = std::make_unique<Player>(
      "Alice", b.getStartTile());
```

To enable players' movement, create methods **forward()** and **backward()**.
Every player has a **vitality** value, which tell us about how tired the player is, its default value is **3**.

Players may have different move behaviors:

- **Normal moves** – (the default), vitality is reduced by 1 with each next move. Once the vitality is 0, the move() method must return an exception with a message "I am too tired to move".

- **Unlimited moves** – such a player is never tired and can move endlessly across the board.

Implement method **Player.sleep()** which restores the default vitality value:

```
alice->sleep();
alice->getVitality();  // this should return 3
```

Playing Cards

A playing card (abstract base class **Card**) is an item that can be located at some tile. Players who are located at the same tile, can pick playing cards. There may be several playing cards (not necessarily unique) on the same tile. Use a FIFO-type structure to store the set of cards on every tile. The methods **tile.addCard(Card c)** and **tile.getFirstCard()** must be implemented. Implement method **tile.getCards()** which returns a collection of cards of this tile, preserving their order. A **Card** must be an abstract base class, so that it should not be possible to create instances of this class.

```
std::shared_ptr<Card> pc = std::make_shared<PrizeCard>();
t1->addCard(pc);
t1->addCard(pc);
auto firstCard = t1->getFirstCard();
auto tileCards = t1->getCards();
```

A player can have up to 5 cards at any given point in time. Implement methods:
Player.pickUp() – allows the player to pick up a first card from the current tile. If a player already has 5 cards, the method should throw an exception.
Player.getCards() – returns a collection of cards of the player.
Player.getCardsByName(name) – returns a range of cards with a given name.
Player.getCardByName(name) – returns a card with the specified name.
Player.drop(Card c) – drops the card, the dropped card becomes the last card on the current tile.

```
alice->pickUp();    // picks up a prize card
alice->pickUp();    // picks up a prize card
auto prizeCards = alice->getCardsByName("prize");
auto prizeCard = alice->getCardByName("prize");
auto aliceCards = alice->getCards();
alice->drop(prizeCard);
alice->drop(prizeCards);
```

Implement methods **onPickedUp()** and **onDroped()** in such a way that for every different types of playing cards it is possible to re-define these methods.

There are various kinds of playing cards.

- **ModifierCard** – they have en effect on the player as long as the player carries such a card.
- **ActionCard** – they have a one-time effect on the player. When such a card is picked up, some event happens to the player, and such a card immediately disappears.

Add abstract base classes **ModifierCard** and **ActionCard** to the class hierarchy of playing cards.

Modifier Cards

- **PrizeCard** – gives 1 point to the player.
- **VitalityCard** – as long as the player carries this card, increases the current vitality value by 1, as well as the maximal vitality value by 1. If the player drops the vitality card, its effect vanishes, and the maximum vitality value must be set back to 3.

```
std::shared_ptr<Card> vc = std::make_shared<VitalityCard>();
t2->addCard(vc);
```

```
alice->forward();
alice->getVitality();                          // returns 2
alice->getMaxVitality();                       // returns 3
alice->pickUp();
alice->getVitality();                          // returns 3
alice->getMaxVitality();                       // returns 4
alice->drop(alice->getCardByName("vitality"));
alice->getVitality();                          // returns 3
alice->getMaxVitality();                       // returns 3
```

- **StickyCard –** the card that gives no benefits to the player, but it is impossible to drop such a card – the method **drop()** should throw an exception.
- **DispelCard –** a card allowing a player to enter the **ExitWonderlandTile** and to finish the game. When the player exits the Wonderland, print the amount of **PrizeCard**-s that a player has.
- **RestlessExplorerCard –** a player who picked this card gets never tired. If the player drops this card, its effect disappears.

Action Cards

- **DistractCard –** if a player picks such a card, any other randomly selected card is dropped.
- **RewindCard –** the player who picked this card is moved to the start tile of the game.
- **ZippyCard –** the player who picked this card increases its vitality value by 1.
- **HypnoticCard –** the player who picked this card loses all its vitality.

Create a type of players **FlyingPlayer** who never get tired, but they can have only up to 3 cards.