```cpp
#include "controller.hpp"

int main() {
    Controller game;
    game.startGame();
}


/* ---------------------------------------------------------------------- */

#ifndef LABO3_CONTROLLER_HPP
#define LABO3_CONTROLLER_HPP

#include <string>
#include <list>
#include "Containers/container.hpp"
#include "Containers/bank.hpp"
#include "Characters/person.hpp"
#include "Characters/boy.hpp"
#include "Characters/girl.hpp"
#include "Characters/dependentPerson.hpp"
#include "Characters/driver.hpp"
#include "Characters/thief.hpp"
#include "Containers/boat.hpp"

/**
 * Classe représentant le controller du jeu
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Controller {
public:
    /**
     * Constructeur par défaut de la classe Controller
     */
    Controller();

    /**
     * Opérateur d'affectation de la classe Controller
     */
    void operator=(const Controller&) = delete;

     /**
      * Impossibilité d'utiliser le constructeur de copie
      * @param other
      */
     Controller(const Controller& other) = delete;

    /**
     * destructeur de la classe Controller
     */
    ~Controller();

    /**
     * Méthode permettant d'afficher un message d'erreur
     * @param errorMsg le message d'erreur
     */
    static void showError(const std::string &errorMsg);

    /**
     * Méthode permettant de gérer le début du jeu
     */
    void startGame();
private:

    /**
     * Méthode permettant de lancer la boucle de jeu
     */
    void nextTurn();
```

```
70
71      /**
72       * Crée le bateau, les deux rives ainsi que tous les personnages du jeu.
73       */
74      void initVariables();
75
76      /**
77       * Méthode finissant un tour de jeu
78       */
79      void endTurn();
80
81      /**
82       * Métode permettant d'embarquer un personnage sur le bateau
83       * @param p le personnage à embarquer
84       */
85      void embark(const Person *p);
86
87      /**
88       * Méthode permettant de débarquer un personnage du bateau
89       * @param p le personnage à débarquer
90       */
91      void disembark(const Person *p);
92
93      /**
94       * Méthode permettant de déplacer le bateau de rive
95       */
96      void moveBoat();
97
98      /**
99      * Méthode permettant d'afficher l'état du jeu
100     */
101     void display() const;
102
103     /**
104      * Méthode permettant de parser les commandes du joueur
105      * @param input la commande entrée par le joueur
106      */
107     void parseInput(const std::string& input);
108
109     /**
110      * Permet de remettre le jeu dans son état initiale afin de le recommencer
111      */
112     void reset();
113
114     /**
115      * Méthode permettant de récupérer l'input du joueur
116      * @return input du joueur
117      */
118     void userInput();
119
120     /**
121     * Méthode permettant d'afficher le menu
122     */
123     static void showMenu();
124
125     /**
126      * Méthode permettant d'afficher une ligne du menu
127      * @param command nom de la commande
128      * @param info description de la commande
129      */
130     static void printMenuLine(char command, const std::string &info, const
131     std::string &argument = "");
132
133     /**
134      * Méthode permettant de convertir un string en personne
135      * @param s le string à convertir
136      * @return le personnage correspondant au string
137      */
138     const Person *compareStringToPerson(const std::string &s) const;
```

```
139
140         /**
141          * Méthode permettant de changer un personnage de container
142          * @param p personnage à changer de place
143          * @param toAdd lieu ou ajouter le personnage
144          * @param addFrom lieu ou enlever le personnage
145          */
146         void changeLocation(const Person &p, Container &toAdd, Container &addFrom);
147
148         /**
149          * Méthode permettant de savoir si le jeu est terminé
150          * @return vrai si le jeu est terminé
151          */
152         bool endOfGame() const;
153
154
155         int turn;
156         bool gameRunning;
157         Bank *leftBank, *rightBank;
158         Boat *boat;
159         std::list<const Person *> people;
160         static const std::string ERROR_MESSAGE, SEPARATOR;
161         static const char DISPLAY = 'p', EMBARK = 'e', DISEMBARK = 'd', MOVE = 'm' ,
162         RESET ='r',EXIT = 'q', HELP = 'h';
163     };
164
165
166     #endif //LABO3_CONTROLLER_HPP
167
168     /* ------------------------------------------------------------------------ */
169     #include "controller.hpp"
170     #include <iostream>
171     #include <iomanip>
172
173     const std::string Controller::ERROR_MESSAGE = "Input invalide ! (Tapez \"h\" pour "
174                                                   "obtenir de l'aide) : ";
175     const std::string Controller::SEPARATOR =
176             "--------------------------------------------------------------------------------";
177
178     Controller::Controller() {
179         initVariables();
180     }
181
182     Controller::~Controller() {
183         delete boat;
184         delete leftBank;
185         delete rightBank;
186         for (const Person *p: people) {
187             delete p;
188         }
189     }
190
191     void Controller::initVariables() {
192         Driver *mother = new Driver("mere");
193         Driver *father = new Driver("pere");
194         Boy *paul = new Boy("paul", *father, *mother);
195         Boy *pierre = new Boy("pierre", *father, *mother);
196         Girl *julie = new Girl("julie", *mother, *father);
197         Girl *jeanne = new Girl("jeanne", *mother, *father);
198         Driver *policeman = new Driver("policier");
199         Thief *thief = new Thief("voleur", *policeman);
200         this->people = {mother, father, paul, pierre, julie, jeanne, policeman, thief};
201         turn = 0;
202         leftBank = new Bank("Gauche");
203         rightBank = new Bank("Droite");
204         boat = new Boat(*leftBank);
205         leftBank->addAll(people);
206         gameRunning = true;
207     }
```

```cpp
void Controller::startGame(){
    showMenu();
    display();
    nextTurn();
}

void Controller::showMenu(){
    std::cout << std::endl;
    printMenuLine(DISPLAY, "afficher");
    printMenuLine(EMBARK, "embarquer", " <nom>");
    printMenuLine(DISEMBARK, "debarquer", " <nom>");
    printMenuLine(MOVE, "deplacer bateau");
    printMenuLine(RESET, "reinitialiser");
    printMenuLine(EXIT, "quitter");
    printMenuLine(HELP, "menu");
}

void Controller::nextTurn() {
    while (gameRunning) {
        userInput();
    }
}

void Controller::parseInput(const std::string &input){
    char command;
    const Person *person = nullptr;
    if (input.empty()) {
        showError(ERROR_MESSAGE);
        return;
    }
    if (input.size() > 1 && input[1] == ' ') {
        person = compareStringToPerson(input.substr(2));
    }
    command = input[0];
    switch (command) {
        case DISPLAY:
            display();
            break;
        case EMBARK:
            embark(person);
            break;
        case DISEMBARK:
            disembark(person);
            break;
        case MOVE:
            moveBoat();
            break;
        case RESET:
            reset();
            break;
        case EXIT:
            gameRunning = false;
            break;
        case HELP:
            showMenu();
            break;
        default :
            showError(ERROR_MESSAGE);
    }
}

void Controller::endTurn() {
    ++turn;
    display();
}
```

```cpp
void Controller::display() const {
    std::cout << std::endl << SEPARATOR << std::endl;
    leftBank->toStream(std::cout);
    std::cout << std::endl;
    boat->toStream(std::cout);
    std::cout << std::endl;
    rightBank->toStream(std::cout);
    std::cout << std::endl << SEPARATOR << std::endl;
}

void Controller::reset() {
    rightBank->emptyContainer();
    leftBank->emptyContainer();
    boat->emptyContainer();
    for (const Person *p: people) {
        leftBank->addPerson(*p);
    }
    boat->moveBoat(*leftBank);
    turn = 0;
}

void Controller::userInput(){
    std::string input;
    std::cout << turn << ">";
    getline(std::cin, input);
    parseInput(input);
}

void Controller::printMenuLine(char command, const std::string &info, const
std::string &argument) {
    std::cout << command << " " << std::setw(8) << std::left << argument <<
              ": " << info << " " << argument << std::endl;
}

const Person *Controller::compareStringToPerson(const std::string &s) const {
    for (const Person *p: people) {
        if (p->getName() == s) {
            return p;
        }
    }
    return nullptr;
}

void Controller::embark(const Person *p) {
    if (boat->isFull() || !p) {
        showError("Error: Bateau est plein ou la personne n'a pas ete trouvee");
    } else if (boat->isOnBank(*p)) {
        Bank *bank = boat->isDockedOnthisBank(*leftBank) ? leftBank : rightBank;
        changeLocation(*p, *boat, *bank);
    } else {
        showError("Error: La personne n'est pas sur la rive");
    }
}

void Controller::disembark(const Person *p) {
    if (boat->isEmpty()) {
        showError("Error: Le bateau est deja vide");
    } else if (boat->isMember(*p)) {
        Bank *bank = boat->isDockedOnthisBank(*leftBank) ? leftBank : rightBank;
        changeLocation(*p, *bank, *boat);
    } else {
        showError("Error: Personne n'est pas dans le bateau");
    }
    endOfGame();
}
```

```cpp
void Controller::changeLocation(const Person &p, Container &toAdd, Container
&toRemove) {
    toAdd.addPerson(p);
    toRemove.removePerson(p);
    if (!(toAdd.isContainerSafe() && toRemove.isContainerSafe())) {
        toAdd.removePerson(p);
        toRemove.addPerson(p);
    } else {
        endTurn();
    }
}

bool Controller::endOfGame() const {
    if (boat->isEmpty() && leftBank->isEmpty()) {
        std::cout << "Vous avez gagne !" << std::endl;
        return true;
    }
    return false;
}

void Controller::showError(const std::string &errorMsg) {
    std::cout << "### " << errorMsg << std::endl;
}

void Controller::moveBoat() {
    Bank* bank = boat->isDockedOnthisBank(*leftBank) ? rightBank : leftBank;
    if(boat->moveBoat(*bank)){
        endTurn();
    } else {
        showError("Il n'y a pas de conducteur dans le bateau");
    }
}

/* ---------------------------------------------------------------------- */

#ifndef LABO3_CONTAINER_HPP
#define LABO3_CONTAINER_HPP

#include <string>
#include <algorithm>
#include <list>

class Person;

/**
 * Classe représentant un conteneur
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Container {
protected:
    /**
    * Constructeur de la classe Container
    * @param name Nom du conteneur
    */
    explicit Container(const std::string &name);

public:
    /**
     * Destructeur de la classe Container
     */
    virtual ~Container() = 0;

    /**
     * Méthode permettant de récupérer le container sous forme affichable
     * @param os opérateur de flux
     * @return le container sous forme affichable
     */
    virtual std::ostream &toStream(std::ostream &os) const;
```

```
415
416         /**
417          * Méthode permettant de récupérer le nom des personnes du conteneur
418          * @return le nom des personnes du conteneur
419          */
420         std::string getPeopleNames() const;
421
422         /**
423          * Méthode permettant de récupérer le nom du conteneur
424          * @return le nom du conteneur
425          */
426         std::string getName() const;
427
428         /**
429          * Méthode permettant de récupérer la taille
430          * @return la taille
431          */
432         unsigned size() const;
433
434         /**
435          * Méthode permettant de vider le container
436          */
437         void emptyContainer();
438
439         /**
440          * Ajoute un personnage au container
441          * @throw std::runtime_error si le container est plein
442          * @param p personnage à ajouter
443          */
444         void addPerson(const Person &p);
445
446         /**
447          * Ajoute plusieurs personnages au container
448          * @throw std::runtime_error si le container est plein
449          * @param people liste de personnages à ajouter
450          */
451         void addAll(const std::list<const Person *> &people);
452
453         /**
454          * Enlève un personne du container
455          * @throw std::runtime_error si le container est vide
456          * @param p personnage à enlever
457          */
458         void removePerson(const Person &p);
459
460         /**
461          * Méthode permettant de savoir si le container est vide
462          * @return true si le container est vide, false sinon
463          */
464         bool isEmpty() const;
465
466         /**
467          * Méthode permettant de savoir si le container est plein
468          * @return true si le container est plein, false sinon
469          */
470         virtual bool isFull() const;
471
472         /**
473          * Méthode permettant de savoir si un personnage est dans le container
474          * @param p personnage à rechercher
475          * @return true si le personnage est dans le container, false sinon
476          */
477         bool isMember(const Person &p) const;
478
479
480
481
482
483
```

```
484       /**
485        * Méthode permettant de savoir si les personnes dans le container sont toutes
486        * en sécurité
487        * @return true si les personnes sont en sécurité, false sinon
488        */
489       bool isContainerSafe() const;
490
491       /**
492        * Méthode retournant un itérateur constant sur le premier élément de la liste
493        * de personnes se trouvant dans le container
494        * @return un itérateur constant sur le premier élément de la liste
495        */
496       std::list<const Person *>::const_iterator begin() const;
497
498       /**
499        * Méthode retournant un itérateur constant sur le dernier élément de la liste
500        * de personnes se trouvant dans le container
501        * @return un itérateur constant sur le dernier élément de la liste
502        */
503       std::list<const Person *>::const_iterator end() const;
504
505   private:
506       const std::string name;
507       std::list<const Person *> people;
508   };
509
510
511   #endif //LABO3_CONTAINER_HPP
512
513   /* ---------------------------------------------------------------------- */
514
515   #include <iostream>
516   #include <sstream>
517   #include "container.hpp"
518   #include "../controller.hpp"
519
520   Container::Container(const std::string &name) : name(name) {}
521
522   Container::~Container() = default;
523
524   std::ostream &Container::toStream(std::ostream &os) const {
525       return os << getName() << " : " << getPeopleNames() << " ";
526   }
527
528   std::string Container::getPeopleNames() const {
529       std::stringstream ss;
530       for (auto &person: people) {
531           ss << person->getName() << " ";
532       }
533       return ss.str();
534   }
535
536   std::string Container::getName() const {
537       return name;
538   }
539
540   void Container::emptyContainer() {
541       people.clear();
542   }
543
544   void Container::addPerson(const Person &p) {
545       if (isFull()) {
546           throw std::runtime_error("le container est plein");
547       }
548       people.push_back(&p);
549   }
550
551
552
```

```cpp
void Container::removePerson(const Person &p) {
    if (people.empty()) {
        throw new std::runtime_error("Le container est vide");
    }
    people.remove(&p);
}

bool Container::isMember(const Person &p) const {
    return std::find(people.begin(), people.end(), &p) != people.end();
}

bool Container::isContainerSafe() const {
    for (const Person *p: people) {
        if (!p->isSafe(*this)) {
            Controller::showError(p->getErrorMessage());
            return false;
        }
    }
    return true;
}

bool Container::isEmpty() const {
    return people.empty();
}

bool Container::isFull() const {
    return false;
}

std::list<const Person *>::const_iterator Container::begin() const {
    return people.cbegin();
}

std::list<const Person *>::const_iterator Container::end() const {
    return people.cend();
}

unsigned Container::size() const {
    return people.size();
}

void Container::addAll(const std::list<const Person *> &peopleToAdd) {
    for (const Person *p: peopleToAdd) {
        if (isFull())
            throw std::runtime_error("le container est plein");
        this->people.push_back(p);
    }
}
/* ---------------------------------------------------------------------- */
#ifndef LABO3_BANK_HPP
#define LABO3_BANK_HPP

#include "container.hpp"

/**
 * Classe représentant une rive de la rivière
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Bank : public Container {
public:
    /**
     * Constructeur de la classe Bank
     * @param name nom de la rive
     * @param people personne se trouvant sur la rive
     */
    explicit Bank(const std::string &name);
};
#endif //LABO3_BANK_HPP
```

```

/* ------------------------------------------------------------------------ */

#include "bank.hpp"

Bank::Bank(const std::string &name) : Container(name) {}

/* ------------------------------------------------------------------------ */

#ifndef LABO3_BOAT_HPP
#define LABO3_BOAT_HPP

#include "bank.hpp"
#include "container.hpp"

/**
 * Classe représentant un bateau
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Boat : public Container {
public:
    /**
     * Constructeur de la classe Boat
     * @param current rive sur laquelle le bateau se trouve
     */
    explicit Boat(const Bank &current);

    /**
     * Méthode permettant de déplacer le bateau sur une autre rive
     * @param bank La nouvelle rive
     * @return true si le déplacement a réussi, false sinon
     */
    bool moveBoat(const Bank &bank);

    /**
     * Méthode permettant de savoir sur quelle rive le bateau se trouve
     * @param bank rive à checker
     * @return true si le bateau se trouve sur la rive, false sinon
     */
    bool isDockedOnthisBank(const Bank &bank) const;

    /**
     * Méthode permettant de récupérer le bateau sous forme affichable
     * @param os opérateur de sortie
     * @return le bateau sous forme affichable
     */
    std::ostream &toStream(std::ostream &os) const override;

    /**
     * Méthode permettant de savoir si une personne est dans le bateau
     * @param person Personne à rechercher
     * @return true si la personne est dans le bateau, false sinon
     */
    bool isOnBank(const Person& person) const;

    /**
     * Méthode permettant de savoir si le bateau est rempli
     * @return true si le bateau est rempli, false sinon
     */
    bool isFull() const override;

private:
    const Bank *currentBank;
    static const std::string RIVER;
    static const int MAX_CAPACITY = 2;
};

#endif //LABO3_BOAT_HPP
```

```

/* ---------------------------------------------------------------------- */

#include <iostream>
#include "boat.hpp"
#include "../controller.hpp"

const std::string Boat::RIVER =
        "================================================================================";

Boat::Boat(const Bank &current) : Container("Bateau"), currentBank(&current) {}


std::ostream &Boat::toStream(std::ostream &os) const {
    if (currentBank->getName() == "Droite")
        os << std::endl << RIVER << std::endl;

    os << Container::getName() << " : " << "< " << Container::getPeopleNames() <<
       ">";
    if (currentBank->getName() == "Gauche")
        std::cout << std::endl << RIVER << std::endl;
    return os;
}

bool Boat::moveBoat(const Bank &bank) {
    bool hasDriver = false;
    auto end = Container::end();
    for(auto it = Container::begin(); it != end; ++it) {
        if ((*it)->canDrive()) {
            hasDriver = true;
            break;
        }
    }
    if (!hasDriver) {
        return false;
    }
    currentBank = &bank;
    return true;
}

bool Boat::isDockedOnthisBank(const Bank &bank) const {
    return &bank == currentBank;
}

bool Boat::isFull() const {
    return Container::size() >= MAX_CAPACITY;
}

bool Boat::isOnBank(const Person &person) const {
    return currentBank->isMember(person);
}

/* ---------------------------------------------------------------------- */

#ifndef LABO3_PERSON_HPP
#define LABO3_PERSON_HPP

#include <list>
#include <string>

class Container;
/**
 * Classe représentant une personne
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */


```

```hpp
class Person {
protected:
    /**
    * Constructeur de la classe Person
    * @param name nom de la personne
    */
    explicit Person(const std::string& name);

public:
    /**
     * Destructeur de la classe Person
     */
    virtual ~Person() = 0;

    /**
     * Méthode permettant de savoir si la personne peut conduire
     * @return true si la personne peut conduire, false sinon
     */
    virtual bool canDrive() const;

    /**
     * Méthode permettant de connaitre le nom de la personne
     * @return le nom de la personne
     */
    std::string getName() const;

    /**
     * Méthode permettant de savoir si la personne est en sécurité dans un lieu
     * @param people liste des personnes présentes dans le lieu
     * @return true si la personne est en sécurité, false sinon
     */
    virtual bool isSafe(const Container& container) const;

    /**
     * Méthode permettant d'avoir un message d'erreur lié à la personne
     * @return un message d'erreur
     */
    virtual std::string getErrorMessage() const;

private:
    const std::string name;
    static const std::string ERROR_MESSAGE;
};


#endif //LABO3_PERSON_HPP

/* ---------------------------------------------------------------------- */
#include "person.hpp"

#include "../Containers/container.hpp"

const std::string Person::ERROR_MESSAGE = "aucune erreur possible";

Person::~Person() = default;

Person::Person(const std::string &name) : name(name) {}

bool Person::canDrive() const {
    return false;
}

std::string Person::getName() const {
    return name;
}

bool Person::isSafe(const Container &container) const {
    return true;
}
```

```cpp

std::string Person::getErrorMessage() const {
    return ERROR_MESSAGE;
}


/* ------------------------------------------------------------------- */

#ifndef LABO3_DRIVER_HPP
#define LABO3_DRIVER_HPP

#include "person.hpp"

/**
 * Classe représentant les conducteurs
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Driver : public Person {
public:
    /**
     * Constructeur de la classe Driver
     * @param name nom du conducteur
     */
    explicit Driver(const std::string &name);

    /**
     * Méthode permettant de savoir si la personne peut conduire
     * @return true si la personne peut conduire, false sinon
     */
    bool canDrive() const override;
};


#endif //LABO3_DRIVER_HPP


/* ------------------------------------------------------------------- */

#include "driver.hpp"

Driver::Driver(const std::string &name) : Person(name) {}

bool Driver::canDrive() const {
    return true;
}


/* ------------------------------------------------------------------- */

#ifndef LABO3_THIEF_HPP
#define LABO3_THIEF_HPP


#include "person.hpp"

/**
 * Classe représentant un voleur
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Thief : public Person {
public:
    /**
     * Constructeur de la classe Thief
     * @param name nom du voleur
     * @param goodWith Personne avec qui le voleur peut rester
     */
    Thief(const std::string &name, const Person &goodWith);
```

```
898
899         /**
900          * Méthode permettant de savoir si la personne est en sécurité dans un lieu
901          * @param people liste des personnes présentes dans le lieu
902          * @return true si la personne est en sécurité, false sinon
903          */
904         bool isSafe(const Container &container) const override;
905
906         /**
907         * Méthode permettant d'avoir un message d'erreur lié à la personne
908         * @return un message d'erreur
909         */
910         std::string getErrorMessage() const override;
911
912     private:
913         static const std::string ERROR_MESSAGE;
914         const Person *dependsOn;
915     };
916
917
918     #endif //LABO3_THIEF_HPP
919
920
921     /* ---------------------------------------------------------------------- */
922
923     #include "thief.hpp"
924     #include "../Containers/container.hpp"
925
926     const std::string Thief::ERROR_MESSAGE = "voleur sans policier";
927
928     Thief::Thief(const std::string &name, const Person &goodWith) : Person(name) {
929         this->dependsOn = &goodWith;
930     }
931
932     bool Thief::isSafe(const Container &container) const {
933         if (container.size() == 1) {
934             return true;
935         }
936         for (auto it = container.begin(); it != container.end(); ++it) {
937             if (*it == this->dependsOn) {
938                 return true;
939             }
940         }
941         return false;
942     }
943
944     std::string Thief::getErrorMessage() const {
945         return ERROR_MESSAGE;
946     }
947
948
949     /* ---------------------------------------------------------------------- */
950
951     #ifndef LABO3_DEPENDENTPERSON_HPP
952     #define LABO3_DEPENDENTPERSON_HPP
953
954     #include "person.hpp"
955     #include <algorithm>
956
957     /**
958      * Classe représentant une personne dépendante de deux autre
959      * @author Friedli Jonathan
960      * @author Jaquier Alexandre
961      */
962
963
964
965
966
```

```hpp
class DependentPerson : public Person {
protected:
    /**
    * Constructeur de la classe DependentPerson
    * @param name nom de la personne
    * @param dependsOn personne de qui la personne dépend
    * @param badWith personne avec laquelle elle ne peut pas rester
    */
    DependentPerson(const std::string &name, const Person &dependsOn, const Person &
    badWith);

public:
    /**
     * Destructeur de la classe DependentPerson
     */
    ~DependentPerson() = 0;

    /**
     * Méthode permettant de savoir si la personne est en sécurité dans un lieu
     * @param people liste des personnes présentes dans le lieu
     * @return true si la personne est en sécurité, false sinon
     */
    bool isSafe(const Container &container) const override;

private:
    const Person *dependsOn;
    const Person *badWith;
};


#endif //LABO3_DEPENDENTPERSON_HPP


/* ---------------------------------------------------------------------- */

#include "dependentPerson.hpp"
#include "../Containers/container.hpp"

DependentPerson::~DependentPerson() = default;

DependentPerson::DependentPerson(const std::string &name, const Person &dependsOn,
                                 const Person &badWith) : Person(name) {
    this->dependsOn = &dependsOn;
    this->badWith = &badWith;
}

bool DependentPerson::isSafe(const Container &container) const {
    bool isSafe = true;
    for (auto it = container.begin(); it != container.end(); ++it) {
        if (*it == this->dependsOn) {
            isSafe = true;
            break;
        } else if (*it == this->badWith) {
            isSafe = false;
        }
    }
    return isSafe;
}

/* ---------------------------------------------------------------------- */

#ifndef LABO3_GIRL_HPP
#define LABO3_GIRL_HPP

#include "dependentPerson.hpp"
```

```cpp
/**
 * Classe représentant la classe Girl
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Girl : public DependentPerson{
public:

    /**
     * Constructeur de la classe Girl
     * @param name nom de la personne
     * @param dependsOn personne dont la personne dépend
     * @param badWith personne avec laquelle elle ne peut pas rester
     */
    Girl(const std::string& name,const Person& dependsOn, const Person& badWith);

    /**
     * Méthode permettant d'avoir un message d'erreur lié à la personne
     * @return un message d'erreur
     */
    std::string getErrorMessage()const override;
private:
    static const std::string ERROR_MESSAGE;
};


#endif //LABO3_GIRL_HPP


/* ---------------------------------------------------------------------- */

#include "girl.hpp"

const std::string Girl::ERROR_MESSAGE = "fille avec son pere sans sa mere";

Girl::Girl(const std::string &name,const Person &dependsOn,const Person &badWith) :
DependentPerson(name,dependsOn,badWith) {}

std::string Girl::getErrorMessage() const {
    return ERROR_MESSAGE;
}

/* ---------------------------------------------------------------------- */

#ifndef LABO3_BOY_HPP
#define LABO3_BOY_HPP

#include "dependentPerson.hpp"

/**
 * Classe représentant un garcon
 * @author Friedli Jonathan
 * @author Jaquier Alexandre
 */
class Boy : public DependentPerson {
public:
    /**
     * Constructeur de la classe Boy
     * @param name nom de la personne
     * @param dependsOn personne dont il dépend
     * @param badWith personne avec qui il ne peut pas rester
     */
    Boy(const std::string &name, const Person &dependsOn, const Person &badWith);

    /**
     * Méthode permettant d'avoir un message d'erreur lié à la personne
    * @return un message d'erreur
     */
    std::string getErrorMessage() const override;
```

```
1105
1106  private:
1107      static const std::string ERROR_MESSAGE;
1108  };
1109
1110
1111  #endif //LABO3_BOY_HPP
1112
1113
1114  /* ---------------------------------------------------------------------- */
1115
1116  #include "boy.hpp"
1117
1118  const std::string Boy::ERROR_MESSAGE = "garcon avec sa mere sans son pere";
1119
1120  Boy::Boy(const std::string &name, const Person &dependsOn, const Person &badWith) :
1121          DependentPerson(name, dependsOn, badWith) {
1122
1123  }
1124
1125  std::string Boy::getErrorMessage() const {
1126      return ERROR_MESSAGE;
1127  }
```