#include <iomanip>

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

using namespace std;

//login stub interface.

//Technical Services from Package Diagram (logical view)

class Login

{

public:

Login(){}

~Login(){}

virtual bool signIn(string username, string password) = 0;

};

//instance of login stub interface- SimpleLogin

//Technical Services (Log In package)

class SimpleLogin: public Login

{

public:

SimpleLogin(){}

~SimpleLogin(){}

bool signIn(string username, string password)

{

return true;

}

};

//Persistence interface- High Score

//Technical Services - High Score from Package Diagram (logical view)

class HighScore

{

public:

HighScore(int highScore1, int highScore2, int highScore3, int highScore4) : \_highScore1(highScore1), \_highScore2(highScore2), \_highScore3(highScore3), \_highScore4(highScore4)

{}

~HighScore(){}

virtual void compareScore(int score)= 0;

protected:

int \_highScore1;

int \_highScore2;

int \_highScore3;

int \_highScore4;

};

//Persistence class instance of interface- LocalHighScore

//Player Package dependency from highscore Package in Technical Services, defined by //Administrator of Domain layer from Package Diagram (logical view)

class LocalHighScore : public HighScore

{

public:

int highScore1, highScore2, highScore3, highScore4;

//hardcoded high scores for comparison

LocalHighScore(int NhighScore1, int NhighScore2 , int NhighScore3 , int NhighScore4): HighScore(highScore1, highScore2, highScore3, highScore4)

{

highScore1= NhighScore1;

highScore2= NhighScore2;

highScore3= NhighScore3;

highScore4= NhighScore4;

//highScore2= 25;

//highScore3= 17;

//highScore4= 12;

cout<< "Created a HighScore Object"<<endl;

}

~LocalHighScore(){}

void compareScore(int score) override

{

cout<< "Comparing against other high Scores..." <<endl;

if(score> highScore1)

{

cout<<" New high score!"<<endl;

cout<< "Score of " <<score << "beats high score of " <<highScore1<<endl;

}

else if( score> highScore2)

{

cout<< "New second highest score!" <<endl;

}

else if (score> highScore3)

{

cout<< "New third highest score!" <<endl;

}

else if (score > highScore4)

{

cout<< "New fourth highest score!" <<endl;

}

else

{

cout<< "Score is not a new high score." <<endl;

}

}

};

//Instance of Abstract Factory: Score Factory

class ScoreFactory

{

public:

static ScoreFactory \* createFactory(){//might need body?};

virtual HighScore \* generateScore(int highScore1, int highScore2, int highScore3, int highScore4)= 0;

//{

//return new LocalHighScore(highScore1, highScore2, highScore3, highScore4);

//}

};

//Instance of Factory Pattern: LocalScoreFactory

class LocalScoreFactory: ScoreFactory

{

public:

LocalHighScore \* generateScore(int highScore1, int highScore2, int highScore3, int highScore4) override

{

return new LocalHighScore(highScore1, highScore2, highScore3, highScore4);

}

};

//resource stub interface. Resource does not have any functions, only a constructor and destructor

//Business Logic underlies in Domain layer in Package Diagram. Dependent on Game package in Domain Layer.

class Resource

{

public:

string resourceType;

int amount;

Resource(){}

~Resource(){}

};

//instance of resource interface- SimpleResource. generates a set amount of resources

//Resource package defined by game package in Domain Layer

class SimpleResource : public Resource

{

public:

string resourceType;

int amount;

SimpleResource()

{

resourceType = "wood";

amount = 4;

}

~SimpleResource(){}

};

//Bulding stub class. generates 1 type of building for use

//Building Package of business logic layer defined by Game Package of the Domain Layer

class Building

{

public:

string type;

int pointValue;

int cost;

Building(){}

~Building(){}

virtual void assignPoints()=0;

};

//instance of polymorphism: Different Buildings

// House is an instance of the building class

//Type of bulding instance created by Game Package

class House: public Building

{

public:

House(){}

~House(){}

void assignPoints() override

{

type = "House";

pointValue = 3;

cost = 2;

}

};

// bridge is an instance of the building class

//Type of building instance created by Game Package

class Bridge: public Building

{

public:

Bridge(){}

~Bridge(){}

void assignPoints() override

{

type = "Bridge";

pointValue= 5;

cost = 4;

}

};

//Skyscraper is an instance of the building class

//Type of building instance created by Game Package

class SkyScraper: public Building

{

public:

SkyScraper(){}

~SkyScraper(){}

void assignPoints() override

{

type = "SkyScraper";

pointValue = 15;

cost= 10;

}

};

// School is an instance of the building class

//Type of building instance created by Game Package

class School: public Building

{

public:

School(){}

~School(){}

void assignPoints() override

{

type = "School";

pointValue= 8;

cost = 6;

}

};

//Gameboard class was removed, replaced with the polymorphic Building Interface

//Player interface

//Player Package of Domain layer

class Player

{

//example of high cohesion: each player class is specific to one player

// also example of low coupling: Player only connects to resources and login, does not conslut the building class

public:

int resources;

int buildings;

int points;

string loginInfo;

Player()

{}

~Player()

{}

virtual bool gatherResource() =0;

//updates point totals and building numbers for the player

virtual void buildBuilding( int cost, int pointValue) = 0;

//checks to see if end condition met

virtual bool endGame() = 0;

//checks to see if it is a valid user.

virtual void validate()= 0;

virtual void quitGame(int playerNumber) = 0;

};

//Simple Player class. Has implemented finalized versions of most functions.

//Domain Layer defines player package and its dependency to the login in packages, and high //sore packages of Technical Services layers.

class SimplePlayer: Player

{

//example of high cohesion: each player class is specific to one player

// also example of low coupling: Player only connects to resources and login, does not conslut the building class

public:

int resources;

int buildings;

int points;

string loginInfo;

SimplePlayer()

{

resources = 0;

buildings= 0;

points = 0;

}

~SimplePlayer()

{}

bool gatherResource()

{

char answer;

SimpleResource gatherResource;

cout << "You are about to gather " << gatherResource.amount << " resources" << endl;

cout << "Do you wish to continue with this action or cancel? Y or N\n";

while (true)

{

cin >> answer;

if (toupper(answer) == 'Y')

{

resources += gatherResource.amount;

cout << "Player has gathered " << resources << " total resources." << endl;

return true;

}

if (toupper(answer) == 'N')

{

return false;

}

}

}

//updates point totals and building numbers for the player

//Bulding package instances defined

void buildBuilding( int cost, int pointValue)

{

resources -= cost;

points+= pointValue;

buildings++;

cout<< "Current resources: " << resources <<endl <<"Current Points: " <<points <<endl <<"Current Buildings: " <<buildings<<endl;

}

//checks to see if end condition met

bool endGame()

{

if( buildings >= 5)

{

return true;

}

else

{

return false;

}

}

//checks to see if it is a valid user. Currently, always returns true

void validate()

{

string username, password;

cout<< "Enter username: ";

cin>>username;

cout<< "Enter password: ";

cin>>password;

SimpleLogin newLogin;

bool isValid=newLogin.signIn(username, password);

if(isValid)

{

cout<< "User Validated"<<endl;

}

else

{

cout<< "User not Authenticated"<<endl;

}

}

// informs the player that they have quit the game. destruction of the player class is handled elsewhere due to the need for the player instances to exist

void quitGame(int playerNumber)

{

cout<< "Player Number " << playerNumber<< "Has quit the game." <<endl;

}

};

//Game interface

//Game Package from Domain Layer in Package Diagram(Logical View)

class Game

{

public:

Game(){}

~Game(){}

virtual void tellWinner(int playerNumber)= 0;

virtual void createGame()=0;

};

//fleshed out simple game, a class from the game interface. Is at the top of the hierarchy.

class SimpleGame: Game

{

// instance of Information expert: houses all the information for the system

public:

void tellWinner( int playerNumber)

{

cout<< "Player number " <<playerNumber << " is the Winner!"<<endl;

}

//instance of Controller: handles how the information is changed and manipulated through the //system

// the meat of the program. creates instances of players and plays the actual game calling all //the classes in proper order

//Package Diagram layers applied, used, and either defined or intervened.

//Domain layer has Game packages utilize player packages, and administrator packages to //uphold game state.

//UI layers Command Line package is utilized in all retrieving data scenarios to advance.

//Player Packages dependency on Technical Services packages utilizes command line to //choose an option to play the game.

// As seen on our SAD document under Logical View Section login Requires Name/ID and //passwords. For whenever a player is added to our Player package then a business logic layers //of buildings and high scores are attached to the Domain layers utilized by the player packages, //administrator packages, and defined by the Game package.

void createGame()

{

//instance of creator: creates the player classes

SimplePlayer player1;

SimplePlayer player2;

SimplePlayer player3;

player1.validate();

player2.validate();

player3.validate();

//GameBoard currentGameboard;

//currentGameboard.GenerateBuilding();

bool foundWinner= false;

int currentPlayer =0;

bool firstPlayer = true;

bool secondPlayer = true;

bool thirdPlayer = true;

int totalPlayers= 3;

char input;

// loop that runs the game

while(!foundWinner)

{

// done using innefficent ways, but was fast and accurate.

//UI layer in package Diagram utilizes Domain layer to retrieve action called

if(currentPlayer%3+1 == 1)

{

if(firstPlayer)

{

cout<< "Player " <<currentPlayer%3 +1 << " choose gather resource (g), build building (b),end game (e), or quit game(q): " ;

cin>> input;

if(input == 'g') //

{

if(player1.gatherResource())

{

currentPlayer++;

}

}

else if(input == 'b')

{

cout<< "Player " << currentPlayer%3+1 << " choose SkyScraper (t), School (s), House (h), or Bridge (b): ";

cin>> input;

if( input == 't')

{

SkyScraper newSkyScraper;

newSkyScraper.assignPoints();

if(player1.resources< newSkyScraper.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player1.buildBuilding(newSkyScraper.cost, newSkyScraper.pointValue);

currentPlayer++;

}

}

else if( input == 's')

{

School newSchool;

newSchool.assignPoints();

if(player1.resources< newSchool.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player1.buildBuilding(newSchool.cost, newSchool.pointValue);

currentPlayer++;

}

}

else if( input == 'h')

{

House newHouse;

newHouse.assignPoints();

if(player1.resources< newHouse.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player1.buildBuilding(newHouse.cost, newHouse.pointValue);

currentPlayer++;

}

}

else if( input == 'b') //Business Logic layer under Domain Layer

{

Bridge newBridge;

newBridge.assignPoints();

if(player1.resources< newBridge.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player1.buildBuilding(newBridge.cost, newBridge.pointValue);

currentPlayer++;

}

}

}

else if(input == 'e')

{

bool test= player1.endGame();

if(test)

{

foundWinner=true;

}

else

{

cout<< "You cannot end the game. " <<endl;

}

}

else if (input== 'q') //UI layer

{

player1.quitGame(1);

firstPlayer=false;

currentPlayer++;

}

}

else

{

currentPlayer++;

}

}

else if(currentPlayer%3+1== 2)

{

if(secondPlayer)

{

cout<< "Player " <<currentPlayer%3 +1 << " choose gather resource (g), build building (b),end game (e), or quit game(q): " ;

cin>> input;

if(input == 'g')

{

if(player2.gatherResource())

{

currentPlayer++;

}

}

else if(input == 'b')

{

cout<< "Player " << currentPlayer%3+1 << " choose SkyScraper (t), School (s), House (h), or Bridge (b): ";

cin>> input;

if( input == 't')

{

SkyScraper newSkyScraper;

newSkyScraper.assignPoints();

if(player2.resources< newSkyScraper.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player2.buildBuilding(newSkyScraper.cost, newSkyScraper.pointValue);

currentPlayer++;

}

}

else if( input == 's')

{

School newSchool;

newSchool.assignPoints();

if(player2.resources< newSchool.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player2.buildBuilding(newSchool.cost, newSchool.pointValue);

currentPlayer++;

}

}

else if( input == 'h')

{

House newHouse;

newHouse.assignPoints();

if(player2.resources< newHouse.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player2.buildBuilding(newHouse.cost, newHouse.pointValue);

currentPlayer++;

}

}

else if( input == 'b')

{

Bridge newBridge;

newBridge.assignPoints();

if(player2.resources< newBridge.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player1.buildBuilding(newBridge.cost, newBridge.pointValue);

currentPlayer++;

}

}

}

else if(input == 'e') //Technical Services utilizes High Score Package

{

bool test= player2.endGame();

if(test)

{

foundWinner=true;

}

else

{

cout<< "You cannot end the game. " <<endl;

}

}

else if (input== 'q')

{

player2.quitGame(2);

secondPlayer=false;

}

}

else

{

currentPlayer++;

}

}

else if(currentPlayer%3+1== 3)

{

if(thirdPlayer)

{

cout<< "Player " <<currentPlayer%3 +1 << " choose gather resource (g), build building (b),end game (e), or quit game(q): " ;

cin>> input;

if(input == 'g')

{

if(player3.gatherResource())

{

currentPlayer++;

}

}

else if(input == 'b')

{

cout<< "Player " << currentPlayer%3+1 << " choose SkyScraper (t), School (s), House (h), or Bridge (b): ";

cin>> input;

if( input == 't')

{

SkyScraper newSkyScraper;

newSkyScraper.assignPoints();

if(player3.resources< newSkyScraper.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player3.buildBuilding(newSkyScraper.cost, newSkyScraper.pointValue);

currentPlayer++;

}

}

else if( input == 's')

{

School newSchool;

newSchool.assignPoints();

if(player3.resources< newSchool.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player3.buildBuilding(newSchool.cost, newSchool.pointValue);

currentPlayer++;

}

}

else if( input == 'h')

{

House newHouse;

newHouse.assignPoints();

if(player3.resources< newHouse.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player3.buildBuilding(newHouse.cost, newHouse.pointValue);

currentPlayer++;

}

}

else if( input == 'b')

{

Bridge newBridge;

newBridge.assignPoints();

if(player3.resources< newBridge.cost)

{

cout<< "Unable to build with current resources" <<endl;

}

else

{

player3.buildBuilding(newBridge.cost, newBridge.pointValue);

currentPlayer++;

}

}

}

else if(input == 'e')

{

bool test= player3.endGame();

if(test)

{

foundWinner=true;

}

else

{

cout<< "You cannot end the game. " <<endl;

}

}

else if (input== 'q')

{

player3.quitGame(3);

thirdPlayer=false;

}

}

else

{

currentPlayer++;

}

}

}

//checks to see who the winner is

ScoreFactory \* newFactory = ScoreFactory::createFactory();

HighScore \* newHighScore = newFactory->generateScore(22,18,14,10);

if(player1.points> player2.points)

{

if(player1.points> player3.points)

{

tellWinner(1);

newHighScore->compareScore(player1.points);

}

}

else if ( player2.points>player3.points)

{

tellWinner(2);

newHighScore->compareScore(player2.points);

}

else

{

tellWinner(3);

newHighScore->compareScore(player3.points);

}

delete newHighScore;

}

};

//(Package Diagram) Game Package from Domain Layer starts up software

int main()

{

SimpleGame newGame;

newGame.createGame();

}