

INTERNET ACADEMY

Institute of Web Design & Software Services

C#8

インターネット・アカデミー

C# 目次

1. 【練習問題】 釣りゲームを作る
2. 画面作成
3. 各クラスの概要(魚エンティティ、魚リスト、ゲーム、スポット)
4. 事前準備① (Fishクラスの作成と動作確認)
5. 事前準備② (FishListクラスの作成と動作確認)
6. 事前準備③ (Placeクラスの作成と動作確認)
7. ゲームの完成

A faint, light blue world map is visible in the background of the slide, centered behind the text.

1. 【練習問題】釣りゲームを作る

作成するアプリの概要

以下のような釣りゲームを作る

【釣りアプリ】

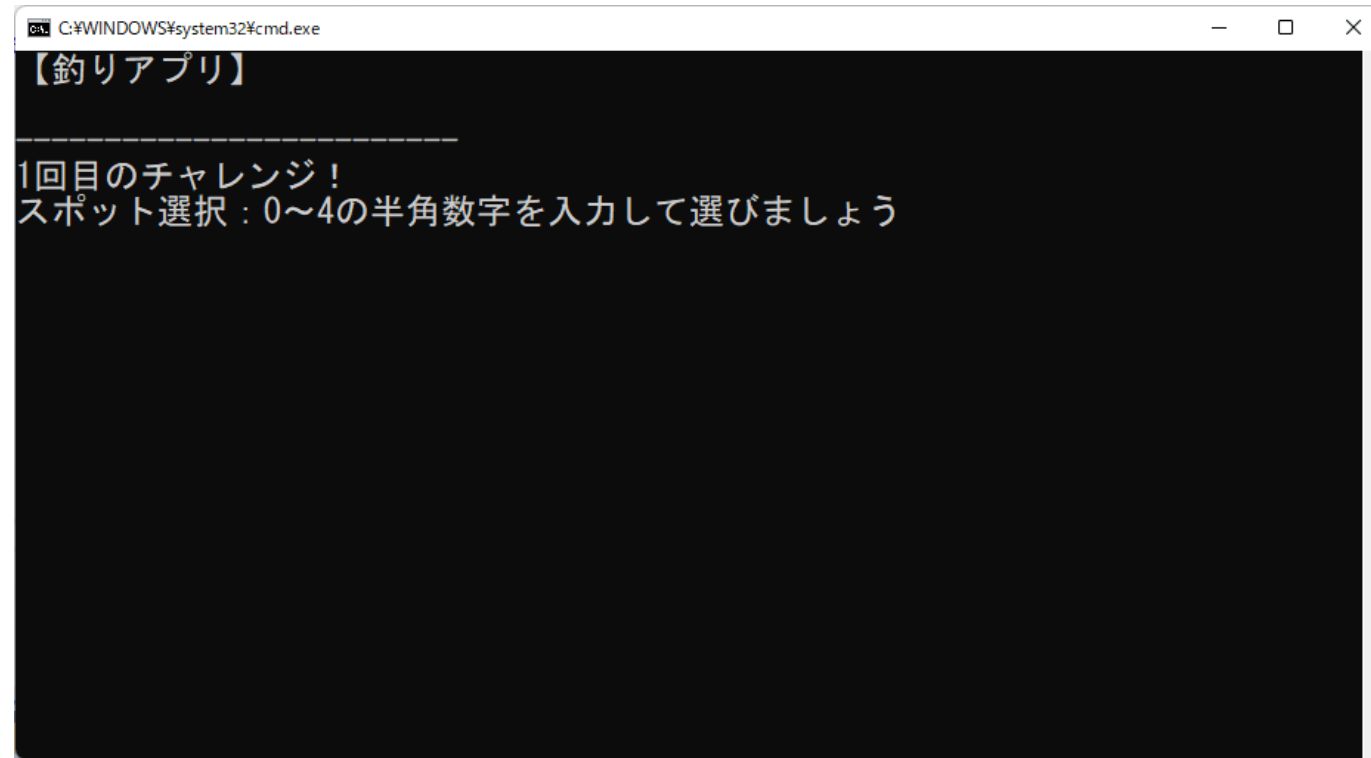
- ・釣れるスポットが5つあり、キーボードからで0～4を入力してもらって釣る
- ・釣れる魚はランダム。10種類あり、魚の種類ごとに獲得ポイントが違う
- ・3回釣った合計点を表示
- ・3回とも同じ場所を選んでも毎回違う魚が釣れるようにする
- ・クラスは4つ利用(Main、Fish、FishList、Place)。さらにクラスを分けてもOK

魚の種類と得点

魚の種類	得点
マグロ	100pt
タイ	120pt
サケ	80pt
ブリ	70pt
イカ	30pt
ホタテ	20pt
アジ	15pt
メダカ	3pt
ゴミ	0pt
おじさん	-50pt

アプリのイメージ

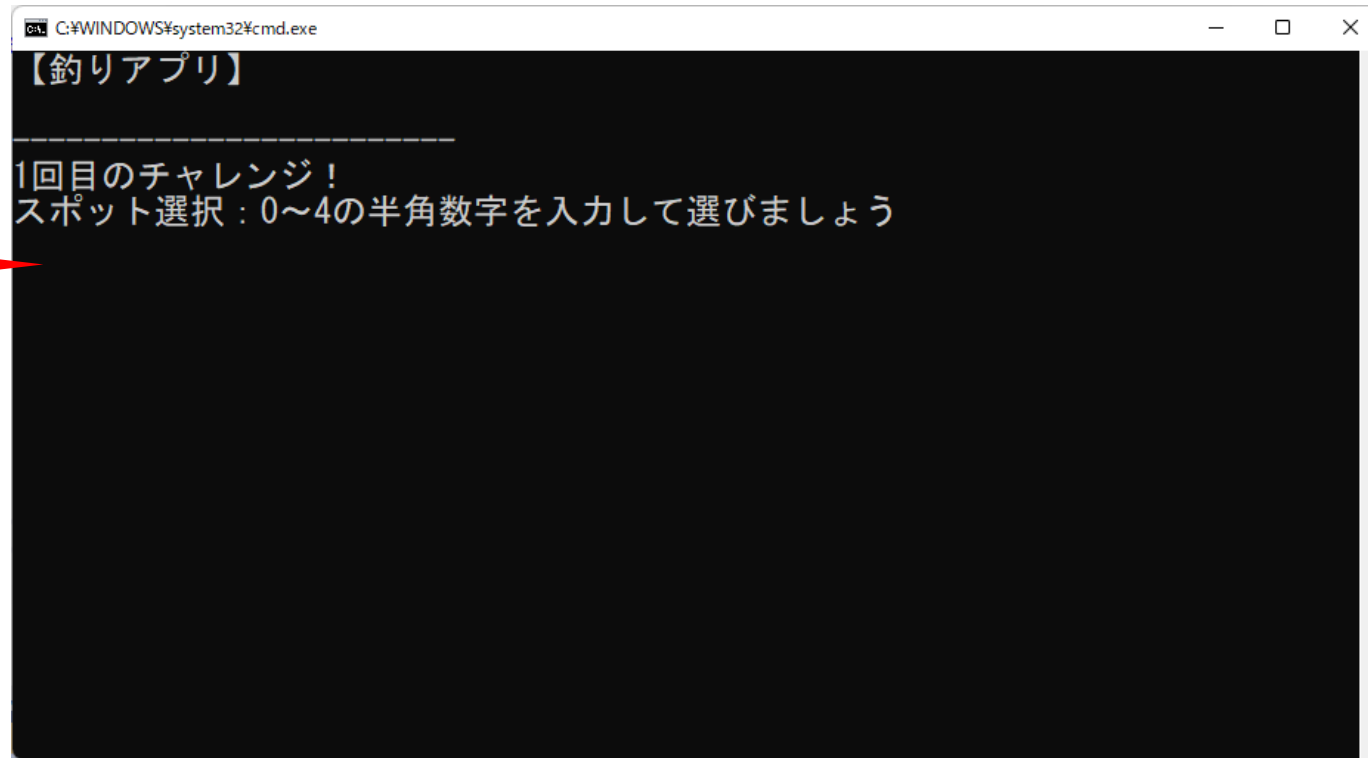
①スタート画面



アプリのイメージ

①スタート画面

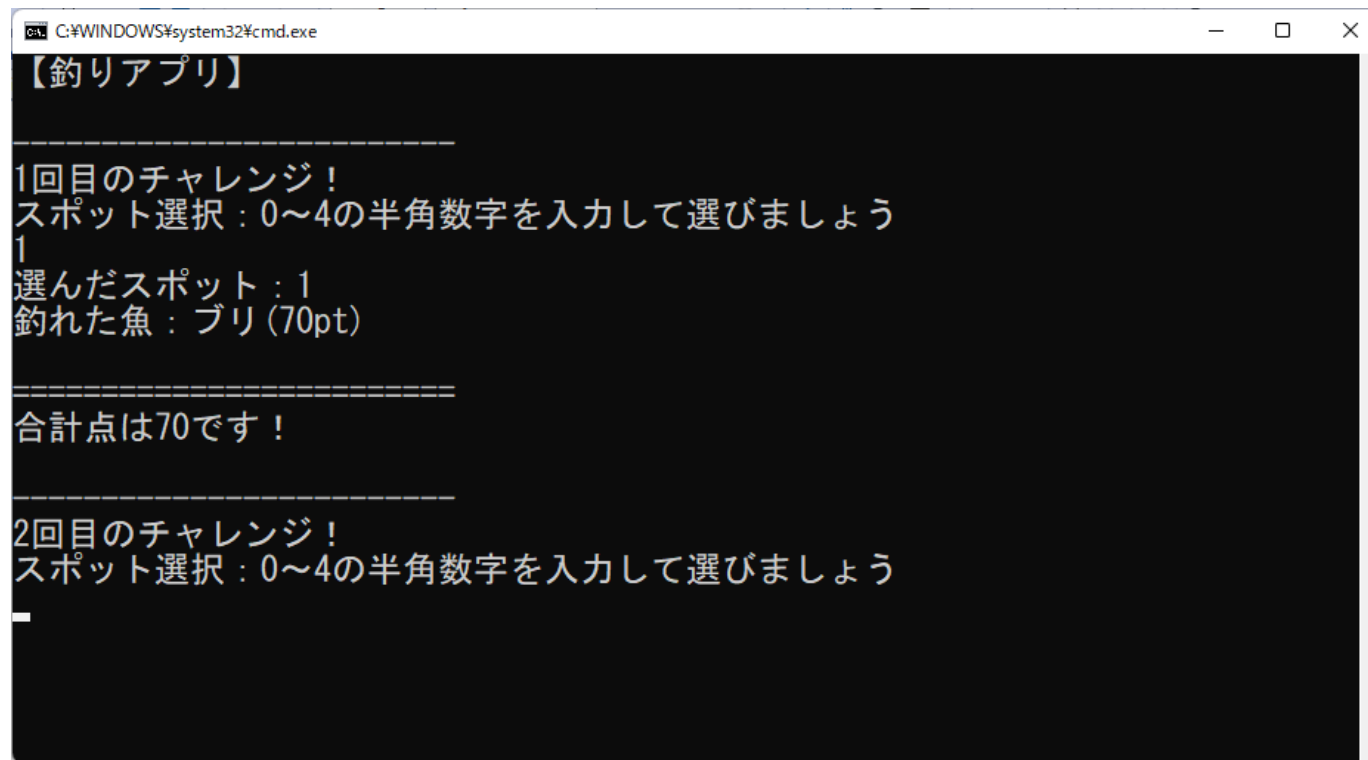
入力待ち状態



```
C:\WINDOWS\system32\cmd.exe
【釣りアプリ】
-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
```


アプリのイメージ

②1回目の入力



```
ca. C:\WINDOWS\system32\cmd.exe
【釣りアプリ】

-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
1
選んだスポット：1
釣れた魚：ブリ(70pt)

=====
合計点は70です！

-----
2回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
_
```

アプリのイメージ

②1回目の入力

数値を入力し
Enterキーを
押す

入力待ち状態

```
C:\WINDOWS\system32\cmd.exe
【釣りアプリ】

-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
1
選んだスポット：1
釣れた魚：ブリ(70pt)

=====
合計点は70です！

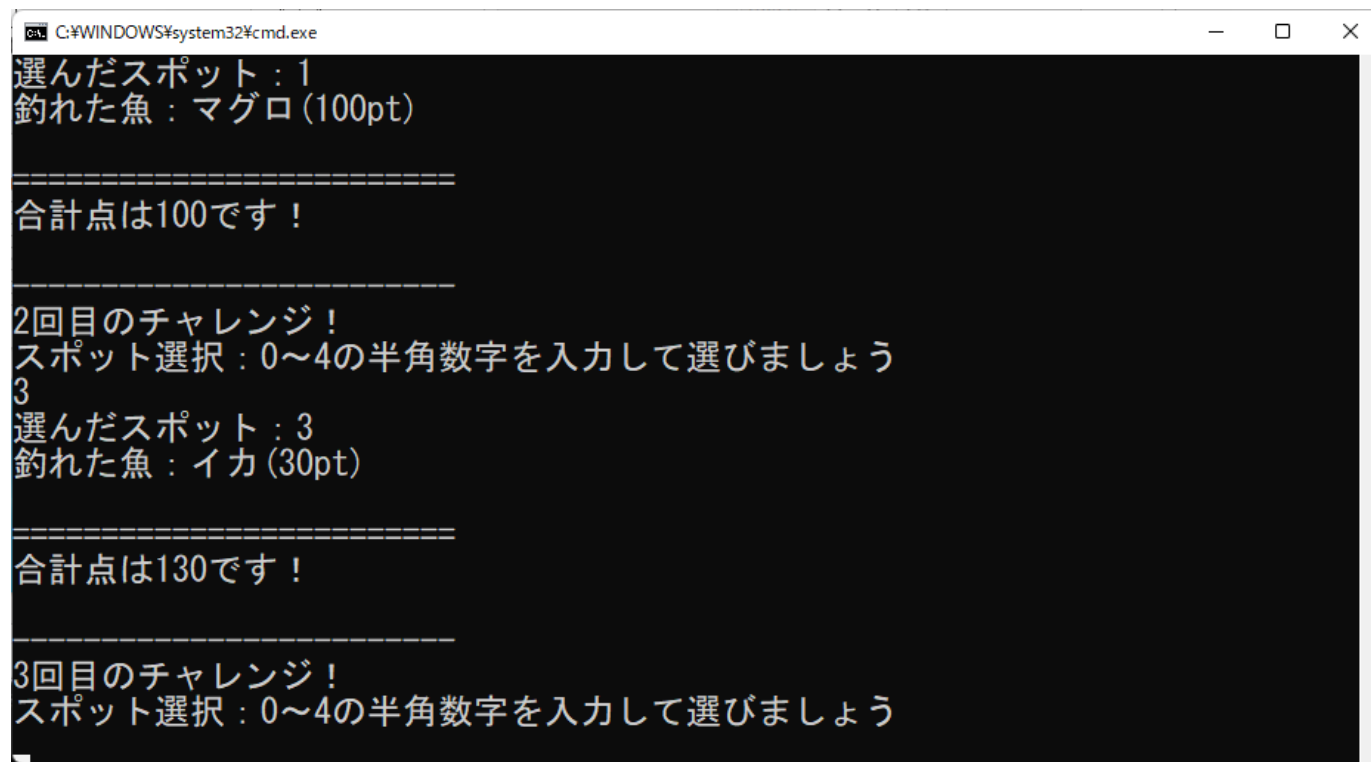
-----
2回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
_
```

魚の得点

合計得点

アプリのイメージ

③2回目の入力



```
C:\WINDOWS\system32\cmd.exe
選んだスポット: 1
釣れた魚: マグロ (100pt)

=====
合計点は100です!

-----
2回目のチャレンジ!
スポット選択: 0~4の半角数字を入力して選びましょう
3
選んだスポット: 3
釣れた魚: イカ (30pt)

=====
合計点は130です!

-----
3回目のチャレンジ!
スポット選択: 0~4の半角数字を入力して選びましょう
```

アプリのイメージ

③3回目の入力 ~ ゲームの終了

```
C:\WINDOWS\system32\cmd.exe

=====
2回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
3
選んだスポット：3
釣れた魚：イカ(30pt)

=====
合計点は130です！

=====
3回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
0
選んだスポット：0
釣れた魚：おじさん(-50pt)

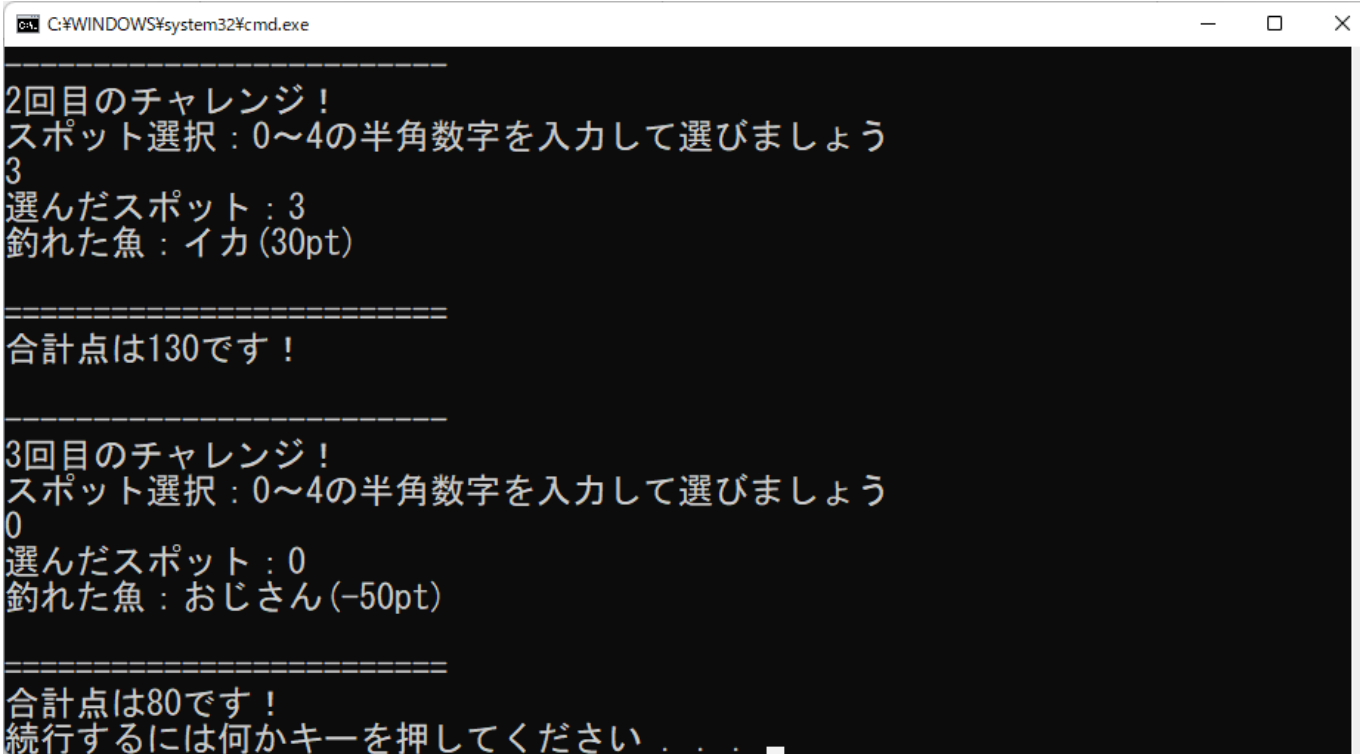
=====
合計点は80です！
続行するには何かキーを押して
```

スコアが減ることもある

合計得点

アプリのイメージ

③3回目の入力 ~ ゲームの終了



```
C:\WINDOWS\system32\cmd.exe

-----
2回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
3
選んだスポット：3
釣れた魚：イカ(30pt)

=====

合計点は130です！

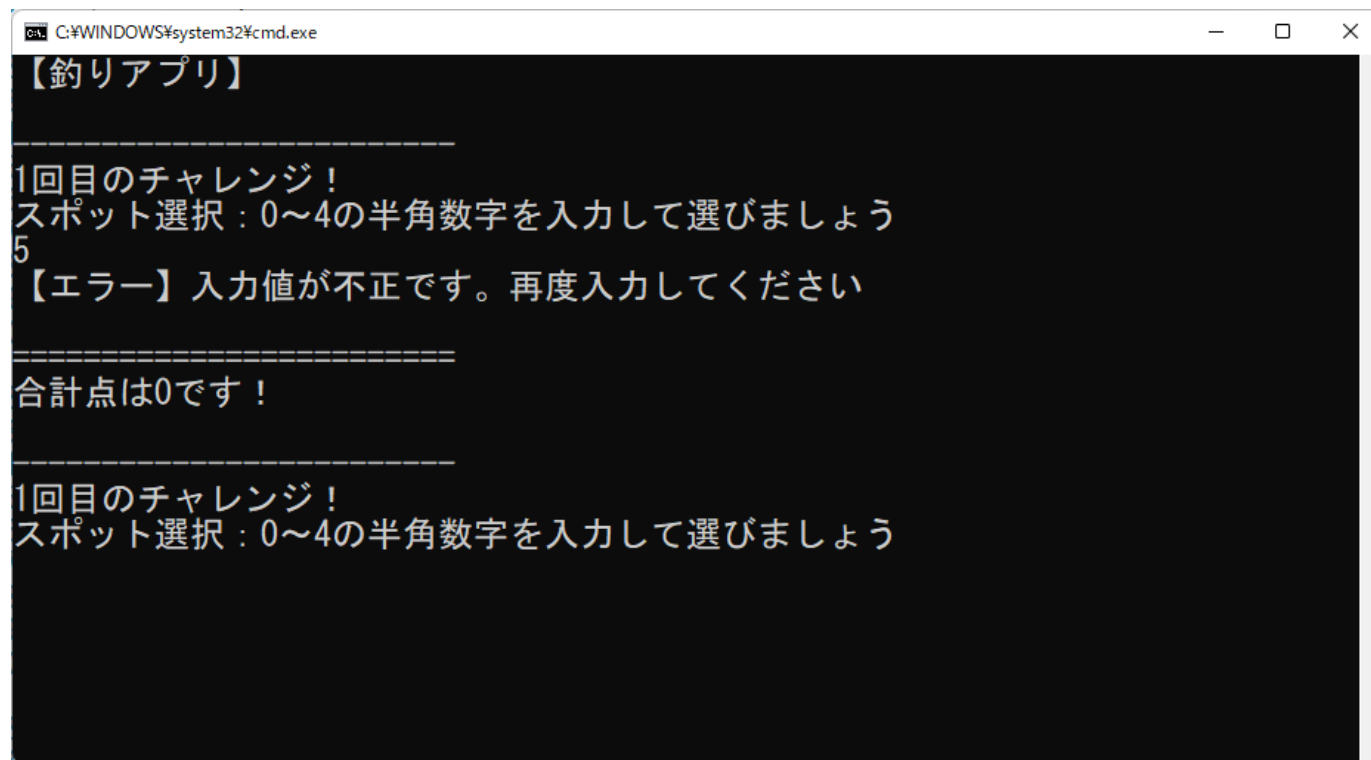
-----
3回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
0
選んだスポット：0
釣れた魚：おじさん(-50pt)

=====

合計点は80です！
続行するには何かキーを押してください . . .
```

アプリのイメージ

範囲外の数字を入力した場合



```
C:\WINDOWS\system32\cmd.exe
【釣りアプリ】

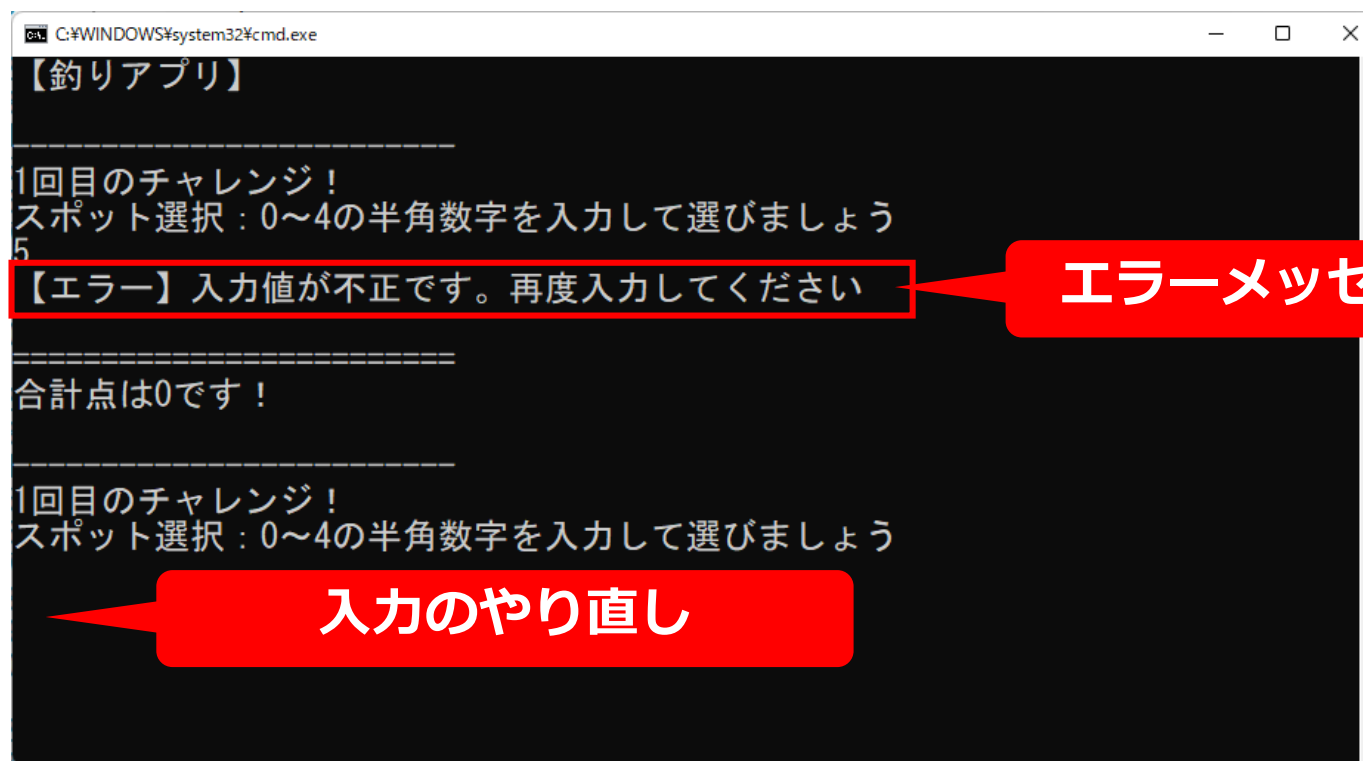
-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
5
【エラー】入力値が不正です。再度入力してください

=====
合計点は0です！

-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
```

アプリのイメージ

範囲外の数字を入力した場合



```
C:\WINDOWS\system32\cmd.exe
【釣りアプリ】

-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
5
【エラー】入力値が不正です。再度入力してください

=====
合計点は0です！

-----
1回目のチャレンジ！
スポット選択：0～4の半角数字を入力して選びましょう
```

エラーメッセージの表示

入力のやり直し



3.各クラスの概要

作成するクラス

作成するクラスは以下のクラス

- ① **Gameクラス** ... ゲーム本体のクラス
- ② **Placeクラス** ... 釣り場のクラス
- ③ **FishListクラス** ... 魚のリストクラス
- ④ **Fishクラス** ... 魚クラス

作成するクラス

①Gameクラス

フィールド

アクセス	型	変数名	初期値	概要
	int	total	0	3回の釣りの合計得点
	int	kaisu	3	釣りをを行う回数

メソッド (public)

戻り値の型	メソッド名	引数	処理内容
-	Game	int kaisu	コンストラクタ (引数をフィールドkaisuに代入)
void	Play	なし	ゲームの本体

メソッド (private)

戻り値の型	メソッド名	引数	処理内容
void	Start	なし	ゲームの開始処理 (メッセージ表示・得点を0に)
void	Finish	なし	ゲームの終了処理 (メッセージ・合計得点表示)

作成するクラス

②Placeクラス

フィールド

アクセス	型	変数名	初期値	概要
public	List<Fish>	pbox	new List<Fish>()	釣り場のリスト

publicなメソッド

戻り値の型	メソッド名	引数	処理内容
-	Place	なし	pboxに5つ分のFishを挿入する
Fish	GetFish	int index	FishListのindex番目のFishを取得する
int	NumOfPlaces	なし	Pboxの数を取得する

作成するクラス

③ FishListクラス

フィールド

アクセス	型	変数名	初期値	概要
private	Random	rnd	new Random()	乱数を発生させるためのメンバ
private	List<Fish>	box	new List<Fish>()	Fishのリスト

メソッド (public)

戻り値の型	メソッド名	引数	処理内容
-	FishList	なし	コンストラクタ (Fishクラスのリストを作る)
Fish	RandomFish	なし	boxの中から1つFishをランダムに選び返す

作成するクラス

④Fishクラス

フィールド

アクセス	型	変数名	初期値	概要
private	string	name	-	魚の名前
private	int	point	-	魚の得点

publicなメソッド

戻り値の型	メソッド名	引数	処理内容
-	Fish	String name int point	コンストラクタ（引数の値を魚のフィールドの魚の名前・得点として設定する）
string	toString	なし	魚の名前・得点の文字列取得（override）

プロパティ

戻り値の型	プロパティ名	該当フィールド	処理内容
int	Point	point	取得専用

作成するクラス

④Fishクラス

フィールド

アクセス	型	変数名	初期値	概要
private	string	name	-	魚の名前
private	int	point	-	魚の得点

publicなメソッド

戻り値の型	メソッド名	引数	処理内容
-	Fish	String name int point	コンストラクタ（引数の値を魚のフィールドの魚の名前・得点として設定する）
string	toString	なし	魚の名前・得点の文字列取得（override）

プロパティ

戻り値の型	プロパティ名	該当フィールド	処理内容
int	Point	point	取得専用

ToStringメソッド

C#のすべてのクラスはObjectクラスを継承している

ToStringメソッドはObjectクラスのオーバーライド可能なメソッド

▼Dummy.cs

```
public class Dummy{  
    public override string ToString(){  
        return "Dummy";  
    }  
}
```

▼Program.cs(Main.csの処理)

```
Dummy d = new Dummy();  
// ToStringメソッドが呼ばれる  
Console.WriteLine(d);
```

▼実行結果

Dummy

Tostringメソッド

C#のすべてのクラスはObjectクラスを継承している

TostringメソッドはObjectクラスのオーバーライド可能なメソッド

▼Dummy.cs

```
public class Dummy{  
    public override string ToString(){  
        return "Dummy";  
    }  
}
```

▼Program.cs(Main.csの処理)

```
Dummy d = new Dummy();  
// ToStringメソッドが呼ばれる  
Console.WriteLine(d);
```

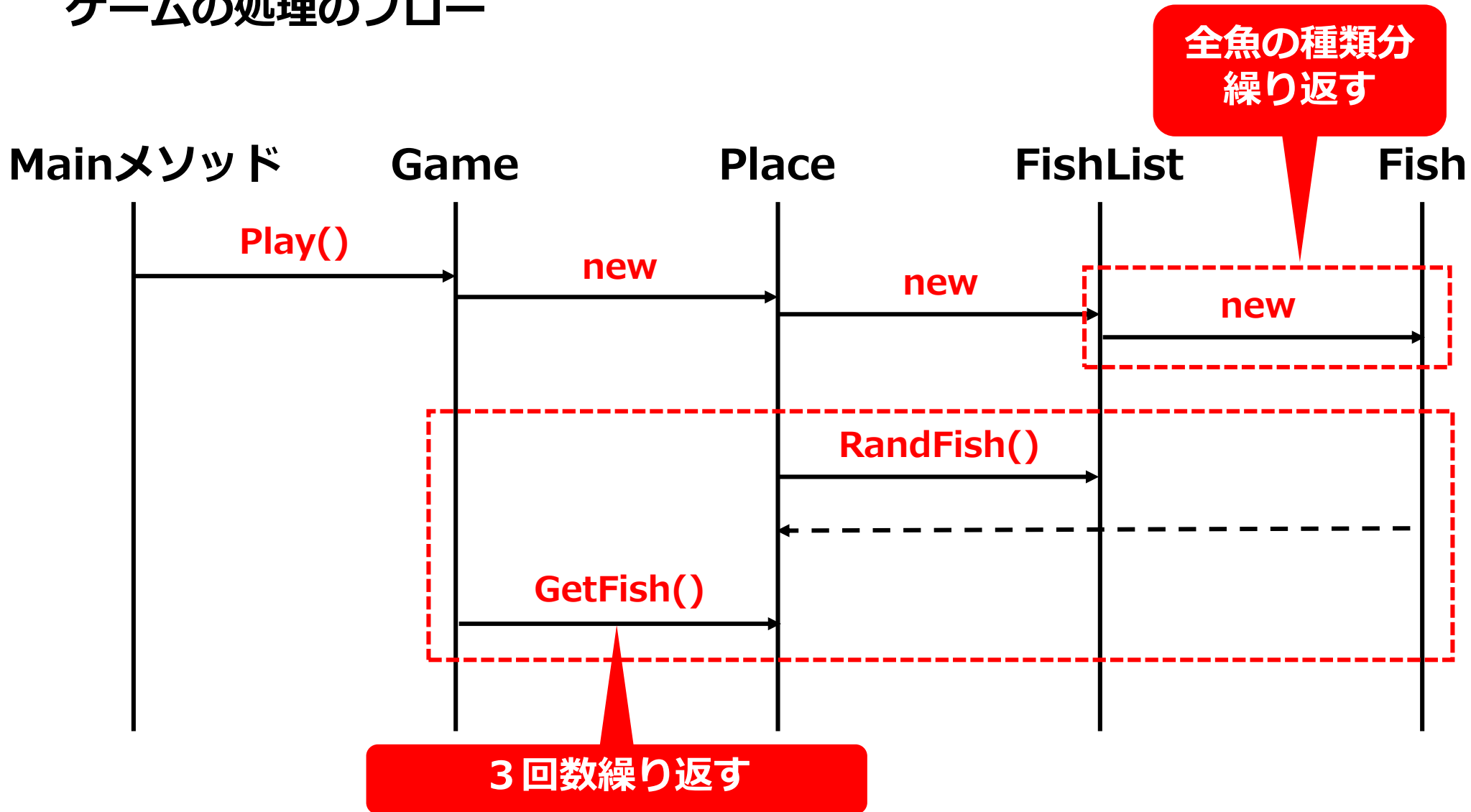
▼実行結果

Dummy

呼び出し

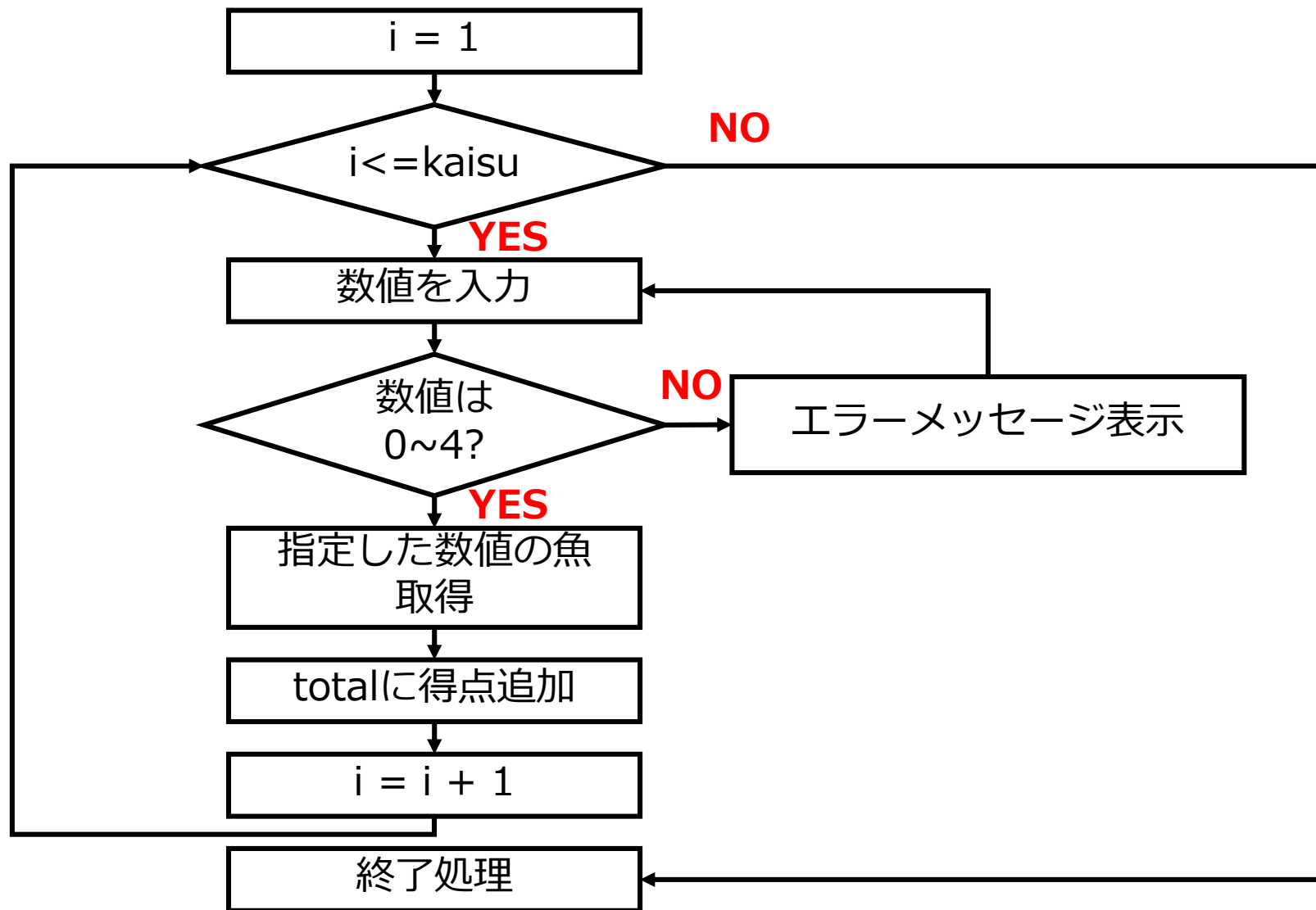
処理のフロー①

ゲームの処理のフロー



処理のフロー②

フローチャート（Playメソッド内）



A faint, light blue world map is visible in the background of the slide, centered behind the main text.

4. 事前準備①

Fishクラスの作成と動作確認

以下の手順でFishクラスを作成し動作を確認する

- ①Fishクラスの実装 … 仕様通りにFishクラスを作成する
- ②Fishクラスのインスタンス生成 … Program.csでインスタンスを生成
- ③Fishクラスの動作確認 … メソッドが仕様通りに動くかを確認する

①Fishクラスの実装

Fishクラス（再掲載）

自分で考えて実装してみましょう

フィールド

アクセス	型	変数名	初期値	概要
private	string	name	-	魚の名前
private	int	point	-	魚の得点

publicなメソッド

戻り値の型	メソッド名	引数	処理内容
-	Fish	String name int point	コンストラクタ（引数の値を魚のフィールドの魚の名前・得点として設定する）
string	toString	なし	魚の名前・得点の文字列取得（override）

プロパティ

戻り値の型	プロパティ名	該当フィールド	処理内容
int	Point	point	取得専用

②Fishクラスのインスタンスの生成

Fishリストの中から任意の魚を選びインスタンスを生成し動作を確認する

記述例：マグロのデータを作って動作を確認する

▼Program.cs(Main.csの処理)

```
// インスタンスの生成（マグロ・100pt）  
Fish fish = new Fish("マグロ", 100);  
  
// ポイントの確認  
Console.WriteLine(fish.Point);  
  
// ToStringの動作確認  
Console.WriteLine(fish);
```

▼実行結果

```
100  
マグロ(100pt)
```

②Fishクラスのインスタンスの生成

Fishリストの中から任意の魚を選びインスタンスを生成し動作を確認する

記述例：マグロのデータを作って動作を確認する

▼Program.cs(Main.csの処理)

```
// インスタンスの生成（マグロ・100pt）  
Fish fish = new Fish("マグロ", 100);  
  
// ポイントの確認  
Console.WriteLine(fish.Point);  
  
// ToStringの動作確認  
Console.WriteLine(fish);
```

▼実行結果

```
100  
マグロ(100pt)
```

ToStringメソッドの実行結果

A faint, light blue world map is visible in the background of the slide, centered behind the title text.

4. 事前準備②

FishListクラスの作成と動作確認

以下の手順でFishListクラスを作成し動作を確認する

- ①FishListクラスの実装 … 仕様通りにFishクラスを作成する
- ②FishListクラスのテストその1 … 魚のリスト一覧が取得できるか確認
- ③FishListクラスのテストその2 … ランダムに魚が取得できるか確認

①FishListクラスの実装

FishListクラス（再掲載）

自分で考えて実装してみましょう

フィールド

アクセス	型	変数名	初期値	概要
private	Random	rnd	new Random()	乱数を発生させるためのメンバ
private	List<Fish>	box	new List<Fish>()	Fishのリスト

メソッド（public）

戻り値の型	メソッド名	引数	処理内容
-	FishList	なし	コンストラクタ（Fishクラスのリストを作る）
Fish	RandomFish	なし	boxの中から1つFishをランダムに選び返す

②FishListクラスのテストその1

FishListクラスにテスト用のクラスメソッドListTestを追加し呼び出してみる

テスト内容：FishListにすべての魚のリストが作成できているか確認する

▼Program.cs(Main.csの処理)

```
FishList.ListTest();
```

作成したFishListクラスに静的なテストメソッドListTestを追加し、呼び出して魚のリストがきちんと作成されているかを確認する

▼実行結果

```
マグロ(100pt)  
タイ(120pt)  
サケ(80pt)  
ブリ(70pt)  
イカ(30pt)  
ホタテ(20pt)  
アジ(15pt)  
メダカ(3pt)  
ゴミ(0pt)  
おじさん(-50pt)
```

②FishListクラスのテストその1

FishListに追加するメソッドは次の通り

FishListメソッド（静的メソッド）

```
public static void ListTest(){  
    FishList fl = new FishList();  
    for (int i = 0; i < fl.box.Count; i++)  
    {  
        Console.WriteLine(fl.box[i]);  
    }  
}
```

FishListクラスのインスタンスを生成し、
boxの中のデータを一つ一つ取得して中身
を確認する

②FishListクラスのテストその2

FishListクラスにテスト用のクラスメソッドRandTestを追加し呼び出してみる

テスト内容：RandFishメソッドでランダムに魚データが取得できるか確認する

▼Program.cs(Main.csの処理)

```
FishList.RandTest(5);
```

作成したFishListクラスに静的なテストメソッドRandTestを追加し、呼び出して引数として与えた数だけランダムに魚のデータが生成されるかを確認する

▼実行結果（ランダムに出力）

```
おじさん(-50pt)  
イカ(30pt)  
メダカ(3pt)  
イカ(30pt)  
ホタテ(20pt)
```

②FishListクラスのテストその2

FishListに追加するメソッドは次の通り

RandTestメソッド（静的メソッド）

```
public static void RandTest(int n)
{
    FishList fl = new FishList();
    for(int i = 0; i < n; i++)
    {
        Console.WriteLine(fl.RandomFish());
    }
}
```

FishListクラスのインスタンスを生成し、
引数nの数だけRandomFishクラスを呼び
出してランダムに魚のデータを出現させる



6. 事前準備③

Placeクラスの作成と動作確認

以下の手順でPlaceクラスを作成し動作を確認する

- ①Placeクラスの実装 … 仕様通りにPlaceクラスを作成する
- ②Placeクラスのテスト … 魚のリスト一覧が取得できるか確認

①Placeクラスの実装

Placeクラス（再掲載）

自分で考えて実装してみましょう

フィールド

アクセス	型	変数名	初期値	概要
public	List<Fish>	pbox	new List<Fish>()	釣り場のリスト

publicなメソッド

戻り値の型	メソッド名	引数	処理内容
-	Place	なし	pboxに5つ分のFishを挿入する
Fish	GetFish	int index	FishListのindex番目のFishを取得する
int	NumOfPlaces	なし	Pboxの数を取得する

②Placeクラスのテスト

Placeクラスにテスト用のクラスメソッドTestを追加し呼び出してみる

テスト内容：FishListにすべての魚のリストが作成できているか確認する

▼Program.cs(Main.csの処理)

```
Place.Test();
```

作成したPlaceクラスに静的なテストメソッドTestを追加し、釣り場が5つできていることと、各釣り場の魚を確認する

▼実行結果

```
場所の数:5  
ホタテ(20pt)  
ホタテ(20pt)  
タイ(120pt)  
イカ(30pt)  
イカ(30pt)
```

②Placeクラスのテスト

Placeに追加するメソッドは次の通り

Testメソッド（静的メソッド）

```
public static void Test() {  
    Place p1 = new Place();  
    Console.WriteLine("場所の数:"+p1.NumOfPlaces());  
    for(int i = 0; i < p1.NumOfPlaces(); i++)  
    {  
        Console.WriteLine(p1.GetFish(i));  
    }  
}
```

Placeクラスのインスタンスを作成し、作成した釣り場の数と各釣り場から魚が取れることを確認する

ゲームの完成

以下の手順でGameクラスを作成しゲームを完成させる

- ①**Gameクラスの実装** … 仕様通りにGameクラスを作成する
- ②**ゲームの完成** … Gameクラスを利用してゲームを完成させる

Gameクラスの実装

Gameクラス（再掲載）

フィールド

アクセス	型	変数名	初期値	概要
	int	total	0	3回の釣りの合計得点
	int	kaisu	3	釣りをを行う回数

メソッド（public）

戻り値の型	メソッド名	引数	処理内容
-	Game	int kaisu	コンストラクタ（引数をフィールドkaisuに代入）
void	Play	なし	ゲームの本体

メソッド（private）

戻り値の型	メソッド名	引数	処理内容
void	Start	なし	ゲームの開始処理（メッセージ表示・得点を0に）
void	Finish	なし	ゲームの終了処理（メッセージ・合計得点表示）

②ゲームの完成

Program.csのメインメソッドでGameクラスのインスタンスを生成

Playメソッドを実行することでゲームを開始できるようにする

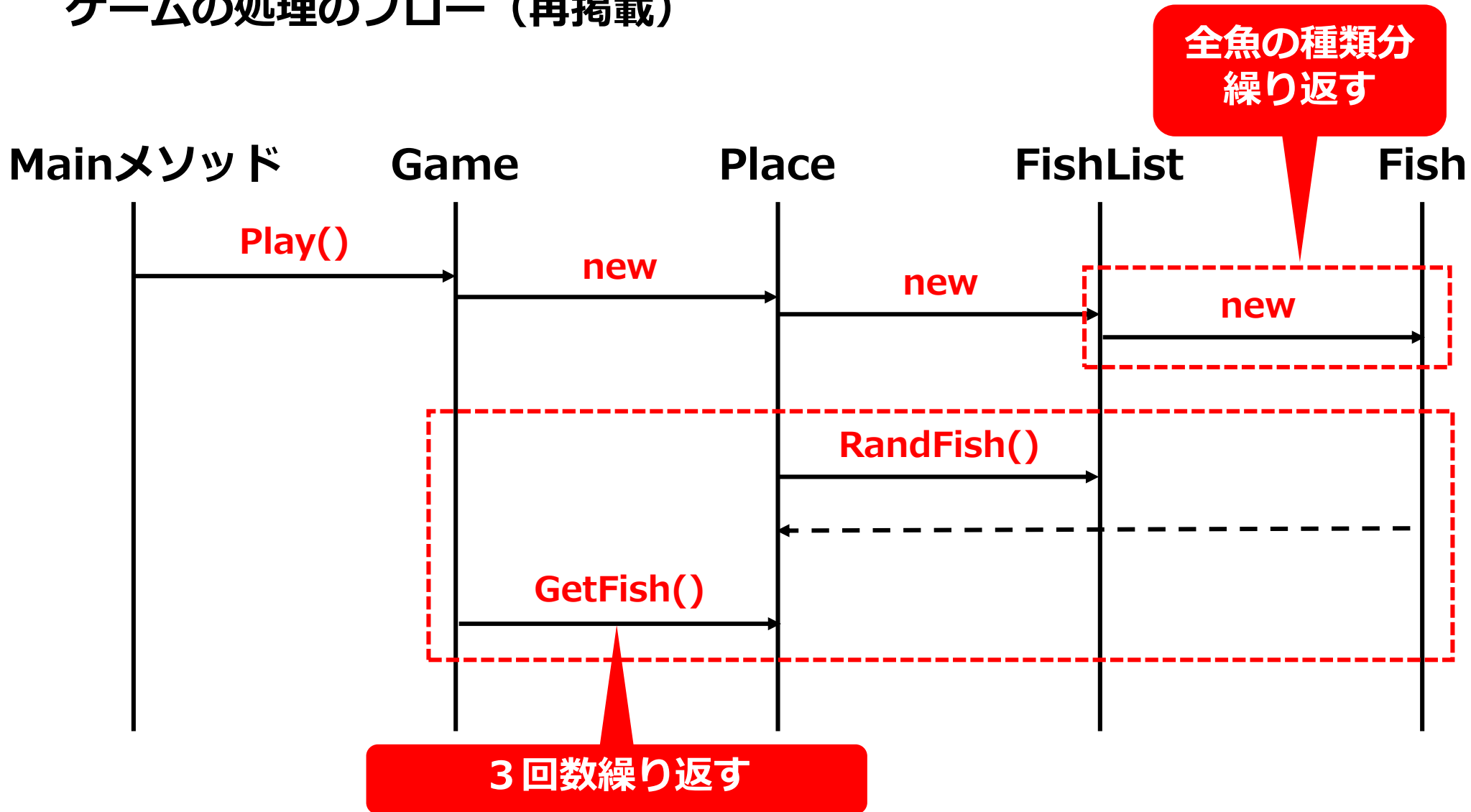
▼Program.cs(Main.csの処理)

```
Game game = new Game(3);  
  
game.Play();
```

Gameクラスのインスタンスを生成し、Playメソッドを実行してゲームがプレイできることを確認する

ゲームの処理のフロー①

ゲームの処理のフロー（再掲載）



ゲームの処理のフロー②

フローチャート（Playメソッド内）（再掲載）

