メモリ管理

C#で正しく学ぶ

2026 改訂版　学習用モデル

この資料で身につけること

- C#の「型」と「置き場所」を分けて考える
- スタックとマネージドヒープの役割を説明できる
- 参照、null、GCの関係を説明できる
- static、構造体、enum、List<T>を正しく捉える

実装の細部を暗記するのではなく、コードの動きを説明する基礎を作ります。

要点：最初の合言葉は「値型＝スタック」と決めつけない

1. メモリの全体像

最初に：これは学習用モデルです

実際の.NETランタイムは、JITやGCなどを含む複雑な仕組みです。

この資料では理解の入口として、次の4つに整理します。

- プログラム領域（命令）
- 静的領域（共有される値）
- ヒープ領域（マネージドヒープ）
- スタック（メソッド呼び出しの作業領域）

要点：実装と完全に一致する「物理的な地図」ではありません

学習用の4領域

領域	入るもの	いつ消えるか
プログラム領域	メソッドの本体（命令）	プログラム終了まで残る
静的領域	staticフィールドの値	プログラム終了まで残る
ヒープ領域（マネージドヒープ）	newした実体	参照がなくなったあと、GCが回収
スタック領域	引数・ローカル変数・戻り先	メソッド終了で消える

要点：「何が入るか」と「いつまで生きるか」をセットで考える

4領域の見取り図

プログラム領域

実行する命令

終了まで残る

静的領域

staticフィールド

終了まで残る

ヒープ領域

マネージドヒープ／newした実体

参照がなくなったあとGC

スタック領域

引数・ローカル変数

メソッド終了で消える

要点：.NET実装ではコードもヒープ系に載ることがある。授業ではこの4つで捉える

プログラム領域（コード領域）

メソッドの本体となる命令を置く、と考えます。

- インスタンスメソッドの命令
- staticメソッドの命令

staticの有無で、コードの置き場は分かれません。

クラスは設計図です。staticなしのclassがプログラム領域、staticありのclassが静的領域、という分け方ではありません。

要点：授業では命令とデータを分けて考える

静的領域

staticフィールドは、インスタンスごとではなく型で共有される値です。

```
class Player
{
    public static int Count;
}
```

値は実行中は保持されますが、終了すれば失われます。次回起動時に自動で残るわけではありません。

要点：staticメソッドの命令は、ここではなくプログラム領域

スタック

メソッド呼び出しの作業領域です。

- 引数
- ローカル変数
- 呼び出し元へ戻るための情報

呼び出しごとにフレームが積まれ、メソッドから戻ると解放されます。LIFO（後から積んだものを先に戻す）です。

サイズは環境や設定で異なるため、「必ず1MB」とは限りません。

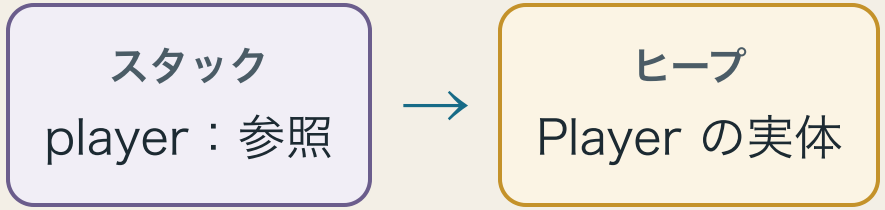
要点：処理終了で消える、とヒープを並べない。消えるのはスタックのフレーム

ヒープ領域（マネージドヒープ）

参照型の変数は、実体への「住所」を保持します。

変数 player：スタック
Player の実体：ヒープ

参照があれば、メソッド終了後も実体は残ります。

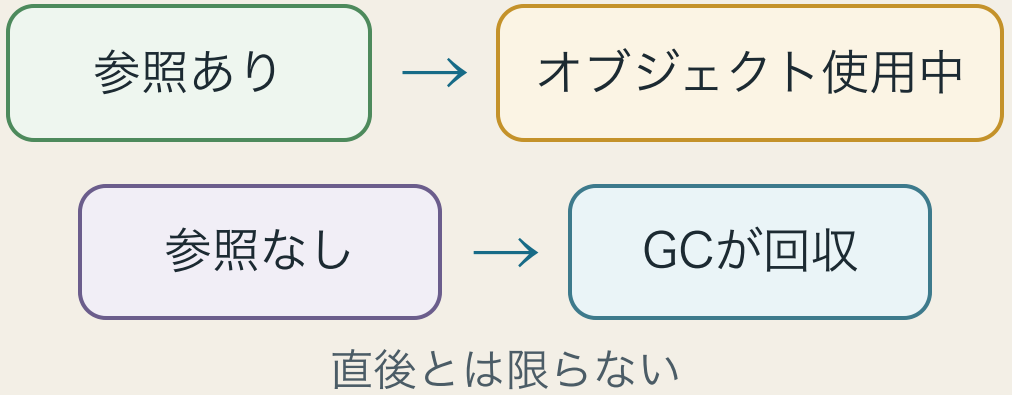


要点：領域の寿命と、オブジェクトの寿命を混同しない

GCとオブジェクトの寿命

GCが回収する対象は、参照されていないオブジェクトです。

参照がなくなっても、すぐ回収されるとは限りません。GCが適切な時点で回収します。



要点：GCの実行タイミングはランタイムが判断する

2. 値型と参照型

型と置き場所は別問題

値型と参照型は、保持するものとコピー方法の分類です。

- 値型：値そのものを保持・コピー
- 参照型：オブジェクトへの参照を保持・コピー

型だけでスタック／ヒープを決められません。配置はローカル、フィールド、配列、ボクシング、最適化などで変わります。

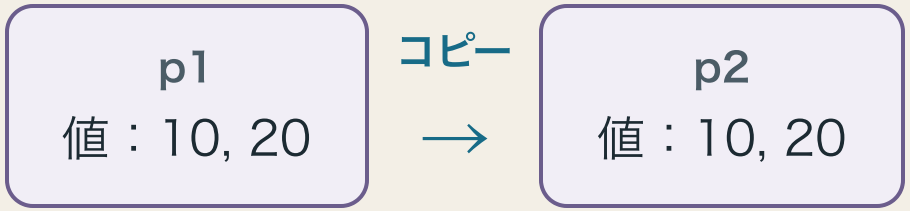
要点：「値型＝スタック、参照型＝ヒープ」と一対一で結び付けない

値型は値をコピー

値型を代入すると、値そのものがコピーされます。

```
Point p2 = p1;
```

p1とp2は別の値です。
片方の変更はもう片方に影響しません。

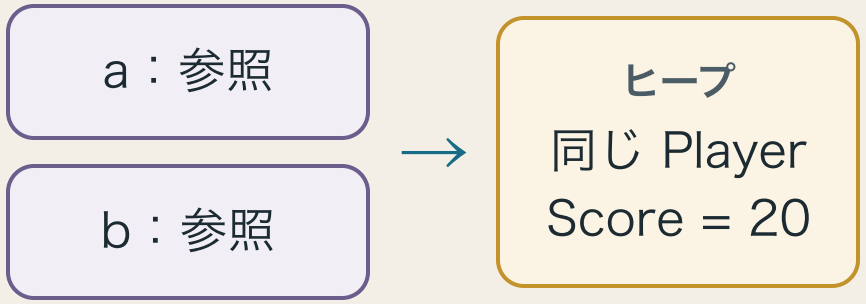


要点：実際の格納位置はJIT最適化で変わることがある

参照をコピーすると

```
Player a = new Player();
Player b = a;
b.Score = 20;
```

コピーされるのはオブジェクト本体ではなく参照です。aとbは同じオブジェクトを指します。



要点：「変数が2つ」でも「実体が2つ」とは限らない

nullと到達可能性

```
player = null;
```

nullは「この変数がオブジェクトを参照していない」という値です。

別の変数やフィールドが同じオブジェクトを参照していれば、オブジェクトはまだ到達可能です。

どこからも到達できなくなった後、GCの回収対象になれます。

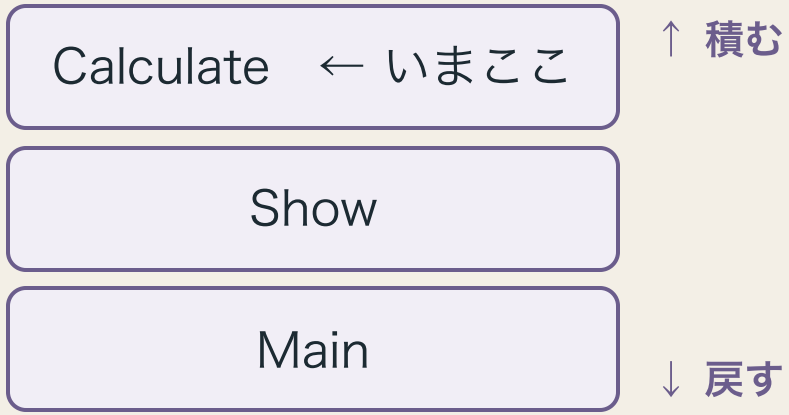
要点：null代入＝即時削除、ではない

メソッド呼び出しとスタックフレーム

メソッドを呼ぶと、その呼び出しの作業領域が積まれます。

Main → Show → Calculate

Calculateが終わる → そのフレームを戻す
Showが終わる → そのフレームを戻す



要点：スタックから参照が消えても、実体の回収はGCの仕事

引数と戻り値

値渡しでは「変数が持っている値」をコピーします。

- 値型の変数：値そのものをコピー
- 参照型の変数：参照をコピー

参照型を値渡ししても、同じオブジェクトを操作できます。ただし引数変数へ別の参照を代入しても、通常は呼び出し元の変数自体は変わりません。

要点：ref / out / in は別の渡し方。この資料では深入りしない

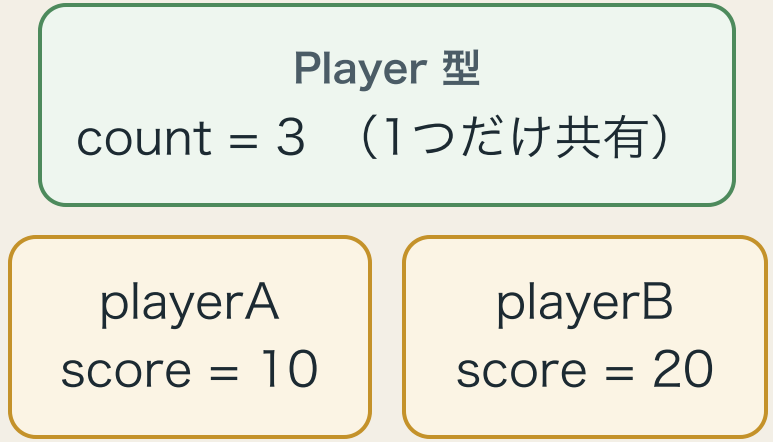
3. static・構造体・enum

staticとインスタンス

staticフィールドは、型ごとに1つの値を共有します。

インスタンスフィールドは、実体ごとに別の値を持ちます。

staticメソッドの命令はプログラム領域です。



要点：アクセス修飾子 public / private もそのまま有効。「誰でも使える」ではない

static初期化の注意

static初期化の時点は、実行条件によって変わります。

初心者向けには、次のように捉えます。

- 型が初めて使われるときに初期化される
- 一度初期化された値は、実行中は共有される
- 終了すれば失われる

要点：「必ず最初のインスタンス作成時」と固定しない

構造体（struct）の基本

構造体は値型です。代入すると通常、値全体をコピーします。

```
Point a = new Point(1, 2);
Point b = a;
b.X = 9;
```

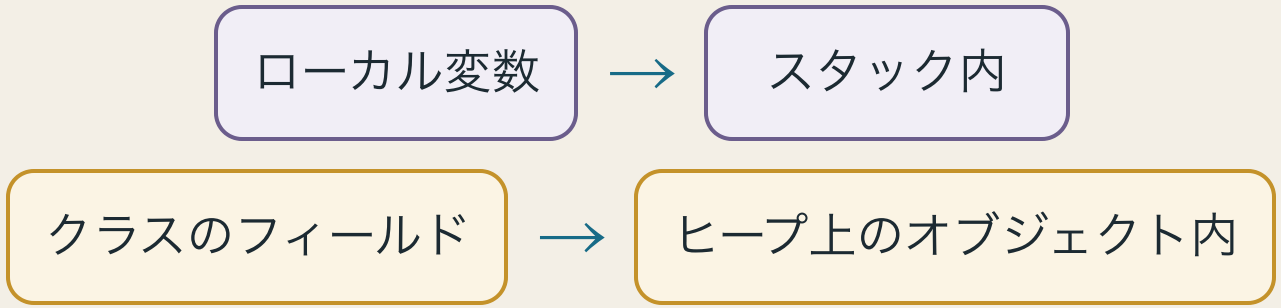
このとき a と b は別の値です。ただし、構造体内に参照型フィールドがあれば、その参照がコピーされます。

要点：値型＝必ずスタック、という意味ではない

構造体はどこに置かれる？

値型でも、必ずスタックに置かれるわけではありません。

- ローカル変数：スタックが一般的
- クラスのフィールド：実体の一部（ヒープ）
- 配列・コレクション：ヒープ上
- ボクシング：ヒープ上



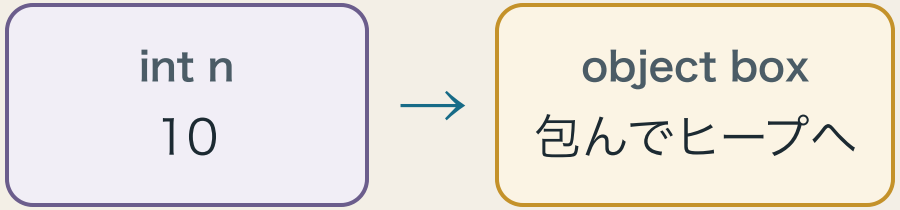
要点：「構造体はスタック上」と断定しない。配置は使われ方で決まる

ボクシング

値型を object として扱うと、値を包む実体が作られます。

```
int n = 10;
object box = n;
```

割り当てとコピーが発生します。



要点：ジェネリックは不要なボクシングを避けやすい

enum（列挙型）

enumは値型で、名前付き定数の集合を表します。配置の考え方は構造体に近いです。

```
enum State : byte
{
    Idle, Running, Paused
}
```

基になる整数型を指定できます。置き場所は、使われる場所に依存します。

要点：値型。置き場所は使われる場所で決まる

4. コレクションと確認

List<T>の内部

List<T>は、内部の配列に要素を保持します。

Count：現在の要素数　　Capacity：配列に入る要素数

Count = 3　Capacity = 6

空きが足りなくなると、内部配列を拡張します。

要点：実装詳細はバージョンにより変わり得る

Count と Capacity

```
var numbers = new List<int>();
numbers.Add(10);
```

- Count：現在入っている要素数
- Capacity：再確保せず格納できる要素数

Count <= Capacity

資料中の「Size」は、List<T>の公開プロパティ名ではありません。

要点：Lengthは配列、CountはList<T>の要素数

newに関する誤解

newは「必ずヒープに置く命令」と覚えないでください。

- クラスのnew：通常、オブジェクトをマネージドヒープへ割り当てる
- 構造体のnew：値を初期化する。配置は文脈次第
- 配列のnew：配列オブジェクトを作る

newの役割は型によって異なります。

要点：構文と物理配置を一対一で結び付けない

実装差と確認の仕方

JITは、不要なローカルを消したり、値をレジスターで扱ったりします。デバッグとリリースでは、観察結果が変わることがあります。

概念図は意味を理解する道具であり、実アドレスの保証ではありません。

確かめるときは https://sharplab.io/

表示は実行環境・最適化で変わります。性能問題は計測して判断します。

要点：図を暗記するより、コードの動きを説明できることが大切

よくある誤解を訂正

- × staticの有無でクラスごと置き場が分かれる
- × staticメソッドは静的データ領域
- × ヒープも処理終了ですぐ消える
- × 値型は必ずスタック
- × nullを代入すると即時に消える
- × static値は次回起動後も残る
- × List<T>の要素数はSize

◯ 型の意味、寿命、配置を分けて説明する

要点：迷ったら「変数が持つもの」「実体」「到達可能性」を確認

まとめ

1. 4領域は学習用モデル
2. 値型／参照型と、スタック／ヒープを同一視しない
3. 参照のコピーでは同じ実体を指す
4. 到達不能になっても、GC回収は後で行われる
5. staticフィールドは型で共有され、終了時に失われる
6. 構造体の配置は使われ方に依存する
7. List<T>は Count と Capacity を分ける

要点：説明できることが、暗記より大切