

---

## ● 授業のポイント

---

- ① Java のメモリ管理について学習します。
  - ② コレクションのメソッドとラムダ式について学習します。
- 

## ● 学習項目

- ・「プログラムコードが実行されるまで（入門編 P. 5）」
- ・「プログラムの実行とメモリ管理（実践編 P. 79）」、スタック、ヒープ
- ・「空きメモリサイズの確認（実践編 P. 81）」
- ・「ガーベッジコレクションとは（実践編 P. 84）」
- ・「forEach メソッドとラムダ式（実践編 P. 138）」
- ・「ラムダ式を用いた並べ替え（実践編 P. 141）」
- ・「sort メソッドによる並べ替え（実践編 P. 118）」
- ・「クラスにつけるアクセス修飾子（実践編 P. 30）」

## ● J2Kad21D 「ヒープ」

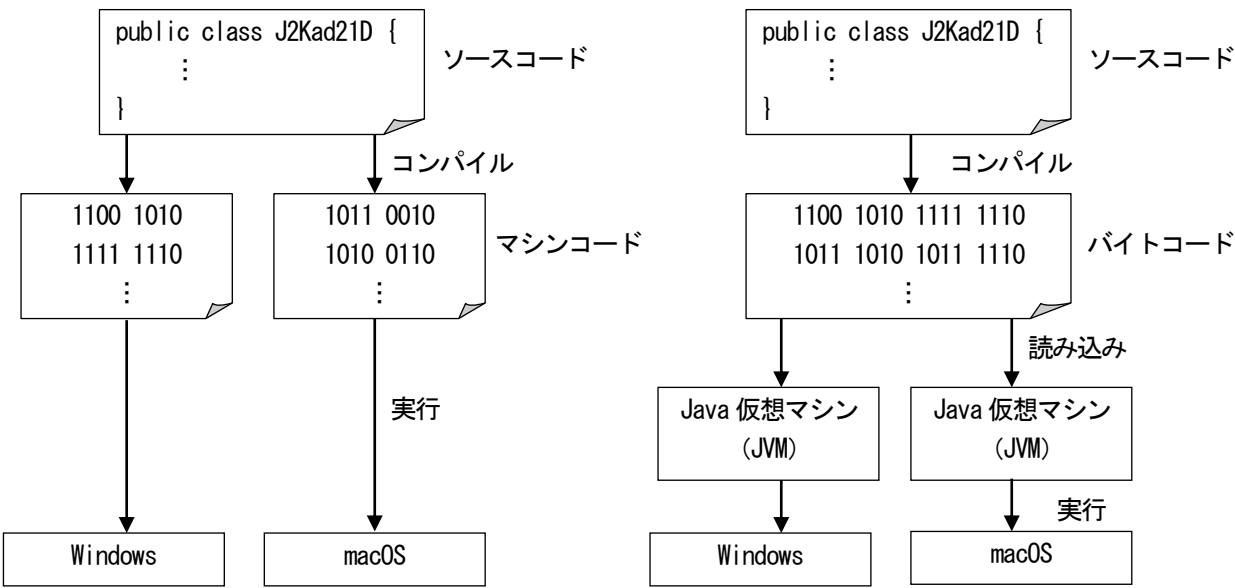
### 0. Java 言語の特徴（入門編 P.5 「プログラムコードが実行されるまで」）

※ ずっとプログラムを作ることを優先してきましたが、このあたりで Java 言語の特徴（C/C++などとの違い）も説明しておいてください（すでに知っているという場合は飛ばして OK です）。

プログラムを実行するには、プログラムが記述されたソースコードをコンパイルしてマシンコードにする必要があります。ただし OS（CPU）ごとに理解できるマシンコードが異なるので、C/C++など昔からある言語では OS ごとにコンパイルを行う必要があります。これに対し Java では OS で直接実行するマシンコードを作るのではなく、Java 仮想マシン（JVM）で実行するためのバイトコードを作ります。あとは各 OS に対応した JVM がバイトコードを読み込んでプログラムを実行します。よって Java では OS が異なっても同じバイトコードで実行できます。

C/C++の場合（OS ごとにコンパイルが必要）

Java の場合（コンパイルは共通、JVM が対応）



1. スタックとヒープ（実践編 P.79「プログラムの実行とメモリ管理」）

プログラム実行時に必要なメモリ領域（変数やインスタンスを作る場所）はスタックとヒープの2種類あります。

種類	用途
スタック	メソッドの呼び出し履歴やメソッド内で使われる変数などで使用。 メソッドが終了するまでの一時的な記憶領域。短期記憶のようなもの。
ヒープ	new で生成するインスタンスなどで使用。 メソッドが終了してもそのインスタンスに対する参照が残っている間は保持される。長期記憶のようなもの。

課題では大量のインスタンスの生成前と生成後のヒープ領域のメモリサイズを確認します。

● J2Kad21C「ガーベッジコレクション」

1. ガーベッジコレクションとは（実践編 P.84）

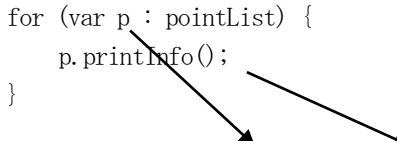
C/C++ではnewで生成したインスタンスは明示的に削除(delete)しない限り、ずっとヒープに残ったままになります。どこからも参照されていない（つまり使っていない）インスタンスも残ったままになり、フラグメンテーションといった不具合につながります。これに対しJavaではインスタンスを削除しなくても、どこからも参照されていないインスタンスを自動的に削除するしくみがあります。これをガーベッジコレクション（ゴミ整理）と呼びます。課題ではガーベッジコレクションの実行を確認します。

## ● J2Kad21B 「コレクションとラムダ式」

### 1. forEach メソッドとラムダ式（実践編 P.138）

コレクション（課題では ArrayList）の forEach メソッドにラムダ式を渡します。最初（パターン A）は拡張 for 文を使って Point クラスの表示→2 倍→表示を行います。このとき拡張 for 文で ArrayList から順次受け取る参照が forEach へ渡すラムダ式の「引数列」、ループ内処理が「処理内容」となります。

```
for (var p : pointList) {  
    p.printInfo();  
}
```



pointList.forEach((引数列) -> {処理内容});      →    pointList.forEach(p -> p.printInfo());

## ● J2Kad21A 「コレクションと並べ替え」

### 1. ラムダ式を用いた並べ替え（実践編 P.141）

コレクション（課題では ArrayList）の sort メソッドを使うとコレクション内のデータの並べ替えを行うことができます。ただし大小比較の方法を sort メソッドに教える必要があり、これをラムダ式で渡します。ラムダ式の引数列は比較する 2 つの Point クラスになります。

(p0, p1) -> {大小比較の方法}

大小比較の結果は int 型で返します（p0 が p1 より小さい場合は負、等しい場合はゼロ、大きい場合は正の整数）。

### 2. sort メソッドによる並べ替え（実践編 P.118）

コレクション内データの並べ替えは Collections クラスの sort メソッドでも行えます。ただしコレクションに格納されているデータ（課題では Point クラス）は Comparable インターフェイスを実装して compareTo メソッドをオーバーライドする必要があります。

```
public int compareTo(Point other)
```

自身（this）と比較相手（other）の大小比較の結果を int 型で返します。

---

## ● J2Kad21S 「シネコン ECC①（ダブルブッキング!）」

### 1. ダブルブッキング

チケット販売の窓口が3つありますが、窓口が異なると同じ番号を発券します。これはそれぞれの窓口が TicketMaker を持っているためです。よって TicketMaker に Singleton を適用すれば、この問題は解決します。基本的に J2Kad17X2 と同じなので、どこに問題があってどういうふうに解決すればいいのか学生に考えさせてください。なお、Singleton は今後の課題でも出てきます。

### 2. クラスにつけるアクセス修飾子（実践編 P.30）

J2Kad21S と J2Kad21X で TicketMaker クラス・Window クラスが異なるため、それぞれ別パッケージ（pac21s と pac21x）に入れています。TicketMaker クラス・Windows クラスの宣言には public をつけていないので各パッケージ内でのみ使うことができます。

## ● J2Kad21X 「シネコン ECC②（スクリーン増設!）」

### 1. TicketMaker の対応

TicketMaker のインスタンスが3つ必要なので、TicketMaker のクラス変数 t を配列にします。あとは getInstance メソッドの引数で TicketMaker のインスタンス番号を受け取るようにし、指定された番号のインスタンスを返すようにすれば OK です。

### 2. 自動券売機

自動券売機 (SalesMachine) クラスを宣言します。さらに SalesPerson クラスと一緒に使えるようにお互いの「親」として TicketSeller インターフェイスを宣言して実装します。あとは main メソッドの SalesPerson 配列を TicketSeller 配列に変更すれば、SalesPerson だけでなく SalesMachine も使うことができます。