



# 発展プログラミング演習II

## 11 ソフトウェアとハードウェア

---

玉田 春昭 水口 充





# ソフトウェアとハードウェア

---

- 空きメモリの確認
- プログラム実行中の状態確認
  - 負荷状態でのプログラムの実行
- 現在時刻の確認



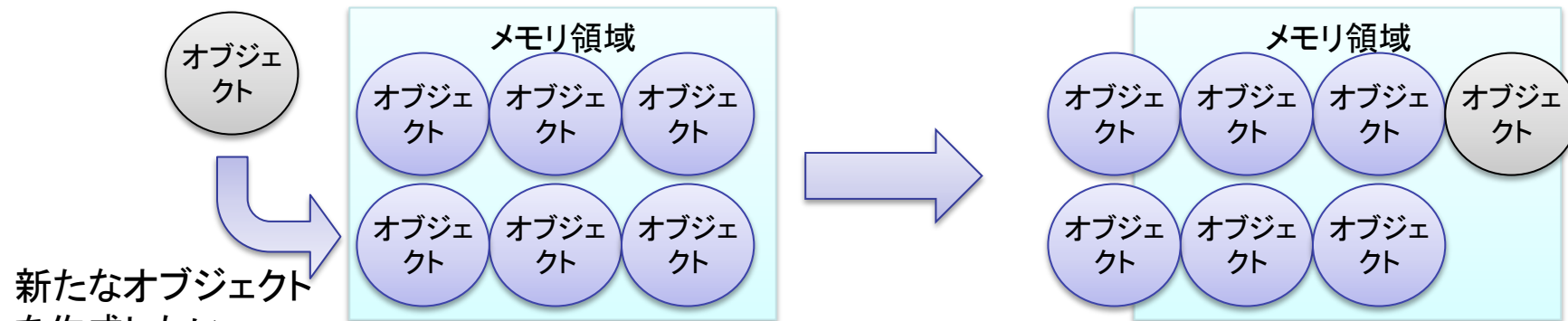
# 空きメモリの確認

---

- `java.lang.Runtime`の3つのメソッドで確認できる.
  - `totalMemory()`
    - 現在のJava仮想マシンへ割り当てられているメモリの総量を返す.
  - `freeMemory()`
    - 現在のJava仮想マシンの空きメモリを返す.
  - `maxMemory()`
    - Java仮想マシンが使用を試みる最大メモリ容量を返す.
- `totalMemory`と`maxMemory`の違いは何か.

# Javaのメモリ管理

- メモリ領域が足りないとき、以下の手順でメモリを確保しようとする。
  - ガベージコレクションを行って、不要なオブジェクトを削除する。
  - それでもメモリ領域が足りなければ、メモリ領域を拡張する。
- これらは自動的に行われるので、プログラミングするときに気にする必要はない。
- `totalMemory`は現在のメモリ割り当て総量を返す。
- `maxMemory`は割り当て可能な最大メモリ量を返す。



2012/12/13

発展プログラミング演習II © 水口・玉田

# 例題11.1

## メモリ使用量の確認

- 右のプログラムを入力, 実行せよ.
  - 空欄には, runtimeを使って, 総メモリ, 空きメモリを取得する.
  - 取得した2つの値をもとに, 使用メモリを計算する.
  - mainメソッドは省略している. 適切な処理を書き加えよ.

```
public class MemoryTest{
    public MemoryTest(){ this(10000); }
    public MemoryTest(int loopCount){
        showMemory();
        heavyProcess(loopCount);
        showMemory();
    }
    private void showMemory(){
        Runtime runtime = Runtime.getRuntime();
        long totalMemory = 
        long freeMemory = 
        long useMemory = 
        System.out.println("総メモリ: " + totalMemory);
        System.out.println("空きメモリ: " + freeMemory);
        System.out.println("使用メモリ: " + useMemory);
        System.out.println(
            "使用率: " + 100d * useMemory / totalMemory);
    }
    private List heavyProcess(int loopCount){
        List list = new ArrayList<>();
        for(int i = 0; i < loopCount; i++){
            list.add("" + i);
        }
        return list;
    }
    // mainは省略.
}
```



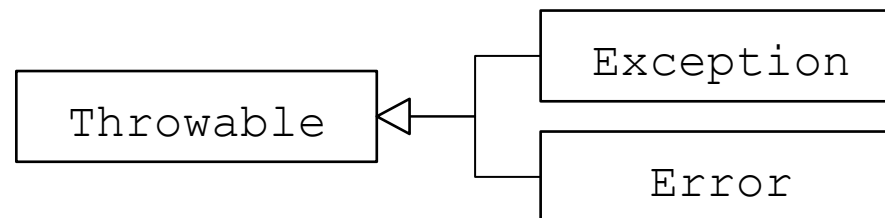
# java.lang.OutOfMemoryError

---

- メモリが足りないときにJVMから投げられる. エラー.
  - `maxMemory`を超えてもメモリを確保しようとしたときに投げられる.
    - `maxMemory`以上はメモリを確保できない.
- いつ起きるかわからない.
  - 不要なオブジェクトを溜めこんでいることが原因のことが多い.
  - 巨大なプログラムで扱うオブジェクトの数が膨大なこともある.

# ErrorとException

- 両方とも`java.lang.Throwable`のサブクラス.
  - `Exception`は回復可能なエラーが発生したことを表す.  
例外.
    - 深刻なエラーではなく, 適切な対応を行えば, プログラムの実行に支障が出ないようなエラー.
    - 例: `NumberFormatException`,  
`ArrayIndexOutOfBoundsException`
  - `Error`は回復不能なエラーが発生したことを表す.
    - プログラムの実行が不可能なほど深刻なエラーが起きた.
    - 例: `OutOfMemoryError`, `ClassFormatError`





# 空きメモリを増やす方法

---

- 不必要なオブジェクトの参照を消す.
  - nullを代入する.
    - どこからも参照されなくなれば, ガベージコレクションの対象になる.
  - 対症療法的. メモリを食いつぶす処理がどこかにあるはず.
- オブジェクトをできるだけ生成しないようにプログラムを変更する.



## 例題11.2

# OutOfMemoryErrorを起こしてみる

- 例題11.1で作成したプログラムのループ回数をどんどん大きくしてみる.
  - 1,000,000を超える辺りからOutOfMemoryErrorが発生する.

## 例題11.3

# 最大メモリ割り当て量の増加

- `OutOfMemoryError`が発生したときのJavaの実行コマンドに以下のオプションを加えて再実行.
  - `-Xmx128m`
    - 最大メモリ割り当て量を128MBにする.
      - `-Xmx256m`だと最大メモリ割り当て量が256MBになる.
      - `-Xmx1024m` までしか指定できない.
    - デフォルトの最大メモリ割り当て量は64MB.
      - 割り当て量を低くすることも可能.
        - » `-Xmx16m`




# ソフトウェアとハードウェア

---

- 空きメモリの確認
- 現在時刻の確認
- プログラム実行中の状態確認
  - 負荷状態でのプログラムの実行

# 現在時間の確認

- `System#currentTimeMillis()`
  - 1970年1月1日0:00:00:00からの経過ミリ秒数をlong型で返す.
- Dateクラス
  - `Date()`
    - 引数なしのコンストラクタでは現在時刻をもとにしたオブジェクトが生成される.
    - `System#currentTimeMillis()` の戻り値を `Date(long)` に渡している.
  - `Date(long)`
    - 引数に与えられた時間をもとにしたオブジェクトが生成される.
- Calendarクラス
  - Dateクラスはタイムゾーンの設定ができない. CalendarはDateに代わり登場した日付を扱うクラス.
    - Dateクラスは日本時間や世界標準時などを扱えない.
  - `Calendar#getInstance()` でオブジェクトを生成する.
  - コンストラクタはprotectedなので, 基本的に呼び出さない.



# 日付のフォーマット (1/2)

---

- `java.text.DateFormat`クラスを用いる.
  - いくつかのstaticメソッドでオブジェクトを作成する.
    - `getDateInstance`
      - 日付のフォーマットのためのオブジェクト.
    - `getTimeInstance`
      - 時間のフォーマットのためのオブジェクト.
    - `getDateTimeInstance`
      - 日付と時間のフォーマットのためのオブジェクト.
  - スタイルは以下の5種類から選択する.
    - `DateFormat#DEFAULT`, `DateFormat#SHORT`,  
`DateFormat#MEDIUM`, `DateFormat#LONG`,  
`DateFormat#FULL`

# 日付のフォーマット (2/2)

- 日付と時間をDEFAULT形式でフォーマットする.


```
Date date = new Date();
DateFormat formatter = DateFormat.getDateInstance(
    DateFormat.DEFAULT, DateFormat.DEFAULT
);
String formattedDateTime = formatter.format(date);
System.out.println(formattedDateTime);
```

- 日付をFULL形式でフォーマットする.

```
Date date = new Date();
DateFormat formatter = DateFormat.getDateInstance(DateFormat.FULL);
String formattedDate = formatter.format(date);
System.out.println(formattedDate);
```

- 時間をMEDIUM形式でフォーマットする.

```
Date date = new Date();
DateFormat formatter = DateFormat.getTimeInstance(DateFormat.MEDIUM);
String formattedDate = formatter.format(date);
System.out.println(formattedDate);
```



# 日付のフォーマット より細やかな指定方法.

---

- `java.text.SimpleDateFormat`を用いる.
  - コンストラクタにフォーマットのパターンを文字列で指定することができる.
  - パターンはAPIドキュメントを参照のこと.



## 例題11.4

# 日付のフォーマット

---

- 日付をDateFormatクラスのオブジェクトを使ってフォーマットして出力せよ.
  - その際, getDateInstanceに以下の5種類の値を渡して得られたオブジェクトでフォーマットせよ.
    - DateFormat#DEFAULT, DateFormat#SHORT, DateFormat#MEDIUM, DateFormat#LONG, DateFormat#FULL
  - クラス名はDateFormatDemoとする.







# jconsole

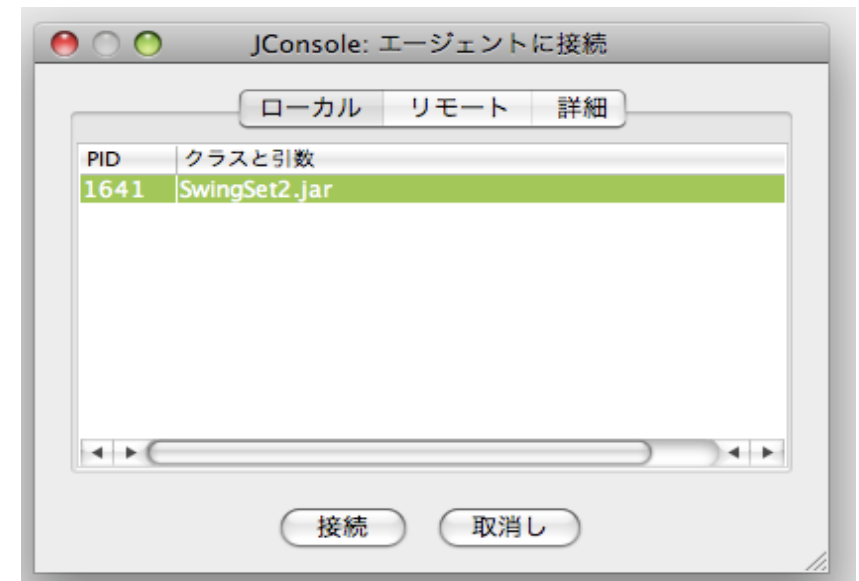
---

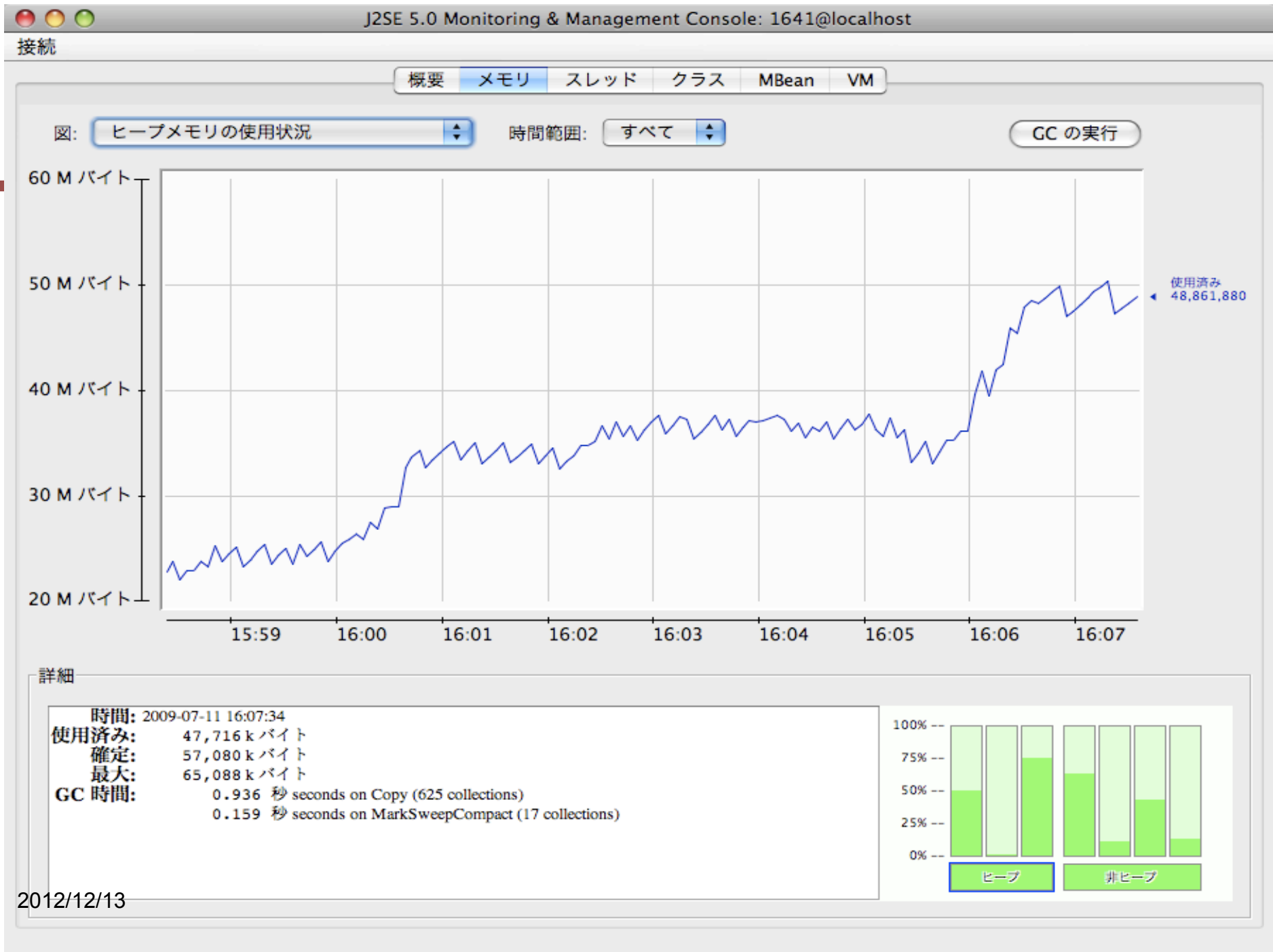
- Java SE 5 以降標準で付属するJMX (Java Management Extensions)準拠のJVM(Java Virtual Machine)監視ツール.
  - Javaアプリケーションのパフォーマンスとリソースの消費に関する情報を提示する.
  - 詳細は以下を参照のこと.
    - <http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/management/jconsole.html>

## 例題11.5

# jconsoleを使う

- jconsoleを起動してSwingSet2の状態を監視してみる.
- SwingSet2の起動
  - `java -jar SwingSet2.jar`
    - `SwingSet2.jar`はMoodleからダウンロードする.
- jconsoleの起動
  - 起動するとダイアログが表示されるので, `SwingSet2.jar`を選択する.







# SwingSet2

---

- JavaのGUIフレームワークの一つであるSwingのデモアプリケーション.
  - 現在SwingSet3まで作成されている.
  - <http://java.net/projects/swingset3>



# ソフトウェアとハードウェア

---

- 空きメモリの確認
- 現在時刻の確認
- プログラム実行中の状態確認
  - 負荷状態でのプログラムの実行



# デバッグ

---

- 一度のコンパイルでプログラムが完成することは、あり得ない。
- コンパイルの失敗や、実行時エラーが、ほぼ必ず起こる。
- コンパイルの失敗
  - エラーメッセージに、エラーの原因と、その箇所（ソースコード中の行番号）が表示される。
  - 当該箇所を確認して修正する。
- 実行時エラー（ランタイムエラー）
  - 実行時に何らかのエラーが発生し、思い通りの結果が得られない。
  - 実行時エラーの原因を特定し、解決することは難しい。

# 代表的なコンパイルエラー

- シンボルを見つけられません.
  - クラスやメソッド, フィールドが見つけれられないときに起こる.
    - つづり間違いや, `import`文がないことが原因のことが多い.
- 式の開始が不明です.
  - 当該個所の前の文が終わっていない.
    - 括弧の対応は合っているか.
- その他
  - `;`がありません.
  - 括弧が閉じられていません
  - :
- 1つの原因により, 大量のエラーが出る場合もある.
  - 一つずつ順番にエラーを修正していくことで, コンパイルは成功する.



# 実行時エラー

---

- まず原因を特定する.
  - 原因は, 変数の値が思い通りになっていない.
    - 数値は思い通りになっているか, 思い通りのクラスのオブジェクトが代入されているか確認する.
- 原因を特定する方法.
  - 変数の値を確認する.
    - `System.out.println`を挿入し, 値をコンソールに出力することで確認する.
    - デバッガを用いる.





# デバッガ

---

- 実行中のプログラムの様子を確認できるツール.
  - 実行の途中(ブレークポイント)で止めて, 変数の値を確認することができる.
- 標準で用意されているデバッガjdbはコマンドライン(CUI)のツールなので, 使いにくい.
- 統合開発環境(IDE)では, 実行箇所がわかるようなデバッガが付属している.
  - Eclipse, NetBeans, ...



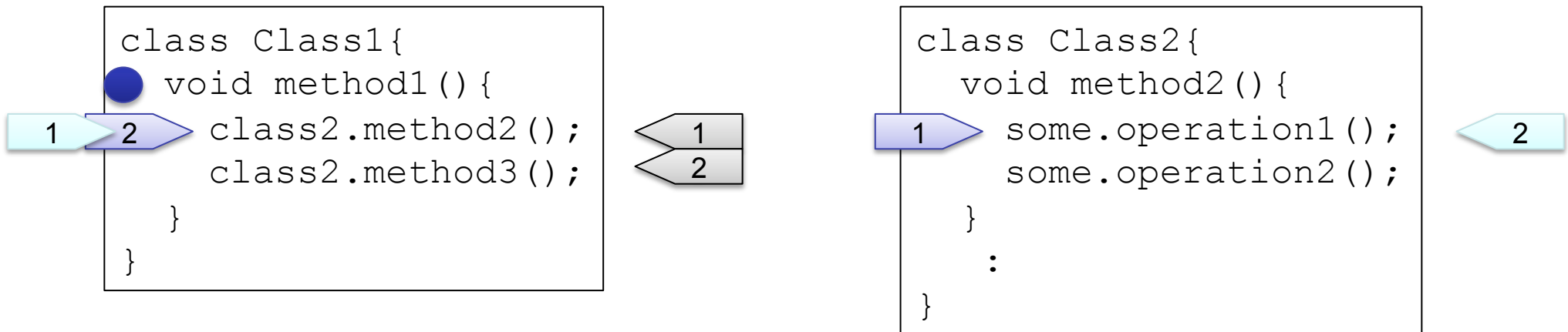
# Eclipseとは

---

- Eclipseは非常によく使われている統合開発環境 (IDE).
  - Eclipseのインストール方法は, Moodleにある「Eclipseのインストール方法 (Mac OS X)」を参照のこと.
  - Eclipseの使い方については, 講義中に説明しない. 以下のURLなどを参照すること.
    - <http://sunjava.seesaa.net/category/3891750-1.html>
    - [http://www.geocities.jp/turtle\\_wide/tools/eclipse/index.htm](http://www.geocities.jp/turtle_wide/tools/eclipse/index.htm)

# デバッグの用語

- ブレークポイント( ● )
  - デバッグ時, プログラムの動作を止める箇所のこと.
  - メソッド名や行番号で指定できる.
- ステップオーバー( ➡ )
  - ブレークポイントで止めたプログラムの実行を1行進める.
- ステップイン( ➡ )
  - ブレークポイントにあるメソッド呼び出しの中にプログラムの実行を移す.
- ステップリターン( ➡ )
  - 現在実行中のメソッドから実行を戻す.





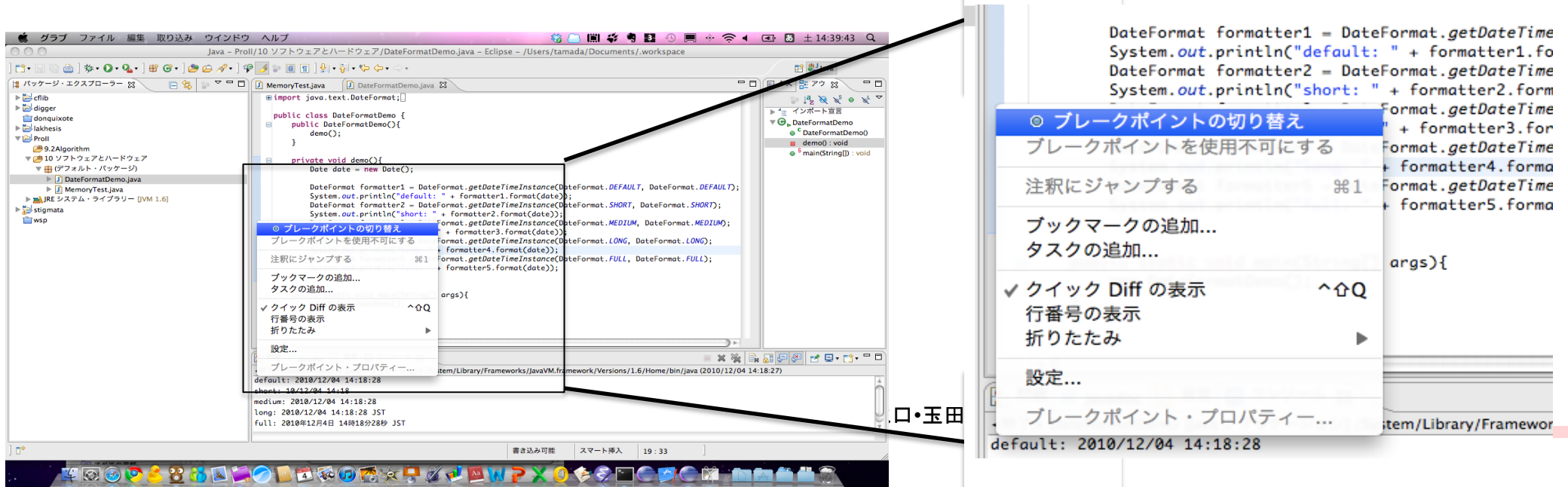
# デバッグの手順

---

- ブレークポイントを設定する.
- プログラムを実行する.
  - ブレークポイントで実行が止まる.
  - 実行中のプログラムの挙動を確かめる.
    - 変数の値が思った通りの値になっているか.
    - 思った通りのメソッドが呼び出されているか.

# Eclipseを用いたデバッグ (1/3)

- ブレークポイントを設定する。
  - エディタ部分の左端(青くなっている箇所)を右クリックすると、メニューが表示される。
  - 「ブレークポイントの切り替え」を選択すると、ブレークポイントが設定されていれば、なしに、設定されていなければブレークポイントが設定される。



# Eclipseを用いたデバッグ (2/3)

- 左端のパッケージエクスプローラから, `main` メソッドのあるクラスのファイルで右クリックする.
  - 出てきたメニューの「デバッグ」→「Javaアプリケーション」を選択する.
    - パースペクティブ切り替えの確認ダイアログが表示されるので, メッセージを確認後, OKをクリックする.
    - 画面が切り替わり, ブレークポイントで実行が止まる.

# Eclipseを用いたデバッグ (3/3)

実行行を操作するためのアイコン

実行している行の時点での変数の値が表示される。

メソッド呼び出しスタック

実行している行が表示される。

画面への出力結果がここに出てくる。

```
import java.text.DateFormat;

public class DateFormatDemo {
    public DateFormatDemo(){
        demo();
    }

    private void demo(){
        Date date = new Date();

        DateFormat formatter1 = DateFormat.getDateInstance(DateFormat.DEFAULT, DateFormat.DEFAULT);
        System.out.println("default: " + formatter1.format(date));
        DateFormat formatter2 = DateFormat.getDateInstance(DateFormat.SHORT, DateFormat.SHORT);
        System.out.println("short: " + formatter2.format(date));
        DateFormat formatter3 = DateFormat.getDateInstance(DateFormat.MEDIUM, DateFormat.MEDIUM);
        System.out.println("medium: " + formatter3.format(date));
        DateFormat formatter4 = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG);
    }
}
```