

オブジェクト指向論(Q)

第10回(OOP4)講義資料

2023/6/12

來村 徳信

今回の講義のテーマと流れ

- オブジェクト指向プログラミング(3) : GUI

- ➡ ● オブジェクト指向の典型的なプログラミングスタイル

- (1) APIのクラスの「拡張」
- (2) イベントに反応するメソッド（イベント駆動型）

- Java API, Swing

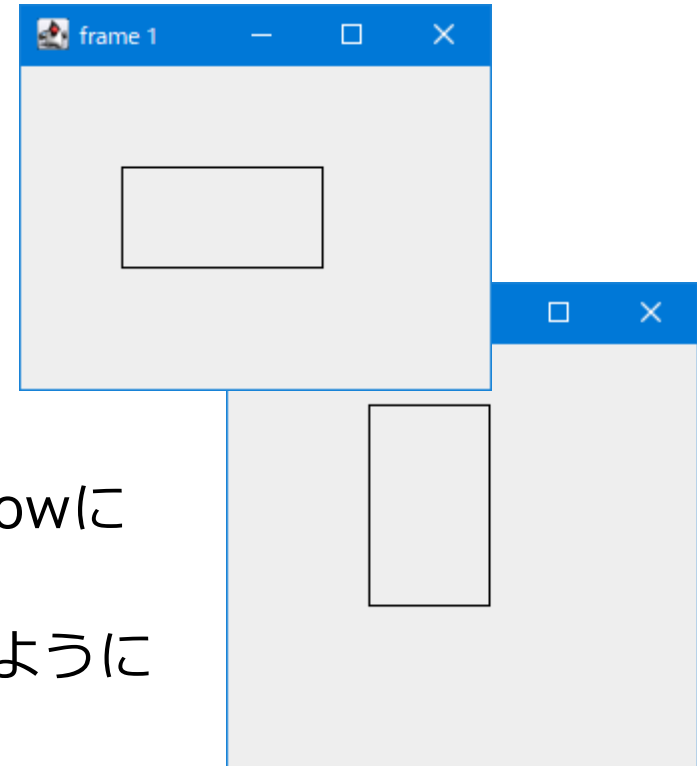
- ウィンドウ

- イベント

- 描画 (paintComponentイベント)

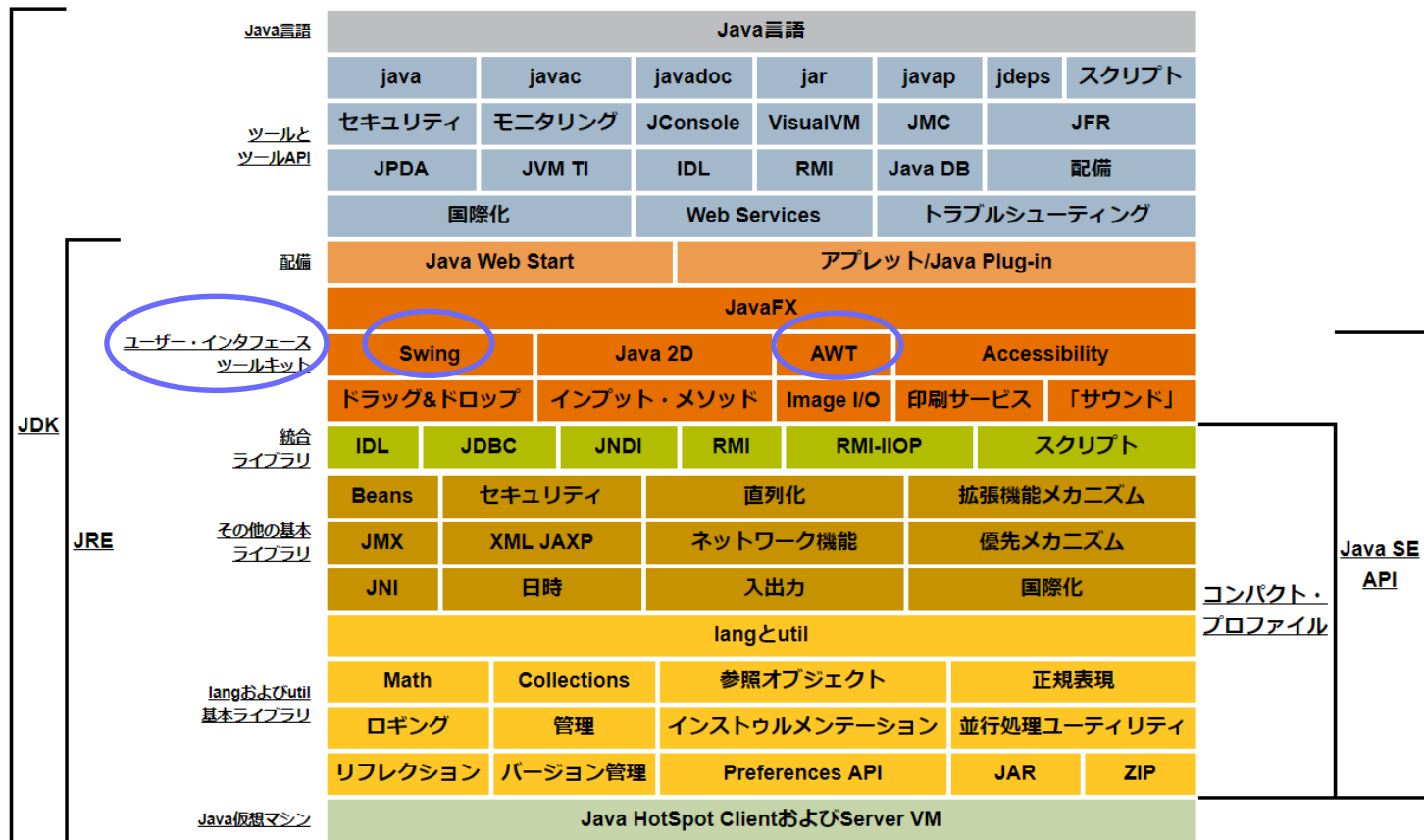
- 例題：図形の描画プログラム

- Window に四角形を描画する
- 最初のステップとして, 1つのWindowに1つずつの四角形
- 来週, 複数の他の図形も描画できるように拡張する.



JavaのAPI

- API (Application Programming Interface)
 - Java言語の（標準的）クラスライブラリのこと
 - 有用なクラスが数多く、パッケージごとに、定義されている
 - 頻繁にバージョンアップされている。本講義は **8** に準拠。



APIのクラスの利用

- APIで定義されているクラスの利用
 - (1) そのクラスを直接使う
 - 例：ArrayList クラス（任意長の配列）
 - Cなどのライブラリと同じ感覚
 - (2) そのクラスを「**拡張**」する
 - オブジェクト指向プログラミング(OOP)の本質
 - 自分で一から作る必要が無い
 - 拡張（**機能追加**）と**カスタマイズ**（標準とは異なる動作）
 - 共通部分はAPIクラスを「**継承**」して用いる.
 - 自分のプログラムに**特有の部分だけ**を「**オーバーライド**」して、記述する.
- 覚える必要はない．必要になったときに調べればよい．
 - 本講義でも，使用するメソッドの**仕様**を与えます．
 - 「どのように**実装**されているか」は知る必要はない．OOPの本質．

Java GUI API : Swing

- Javaの ユーザーインターフェイス API
 - AWT, Swing, JavaFX など複数の API パッケージがある.
 - 本講義では Swing を使う
 - プラットフォーム独立な, Window やボタンなどUIコンポーネントが定義されている.
 - Event などは AWT で定義されている.
 - 古いが, 基本の理解 (↓) に向いている.
- 本講義の狙い (再掲)
 - オブジェクト指向の典型的なプログラミングスタイル
 - (1) APIのクラスの「拡張」
 - (2) イベントに反応するメソッド (イベント駆動型)

イベント駆動型プログラム

● イベント

- ユーザがGUIを操作するとイベントが発生する.
- イベントが発生したということが通知される (メッセージパッシング) ÷ 特定のメソッドが呼ばれる.
- ユーザの操作に限らず, 例えば画面の「再描画要求」などもシステムからプログラムへのメッセージとして送られる.

● オブジェクトがイベントに反応する

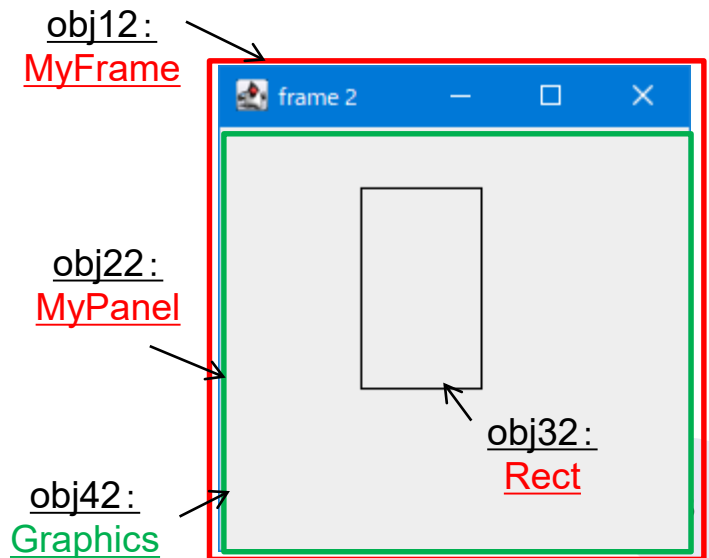
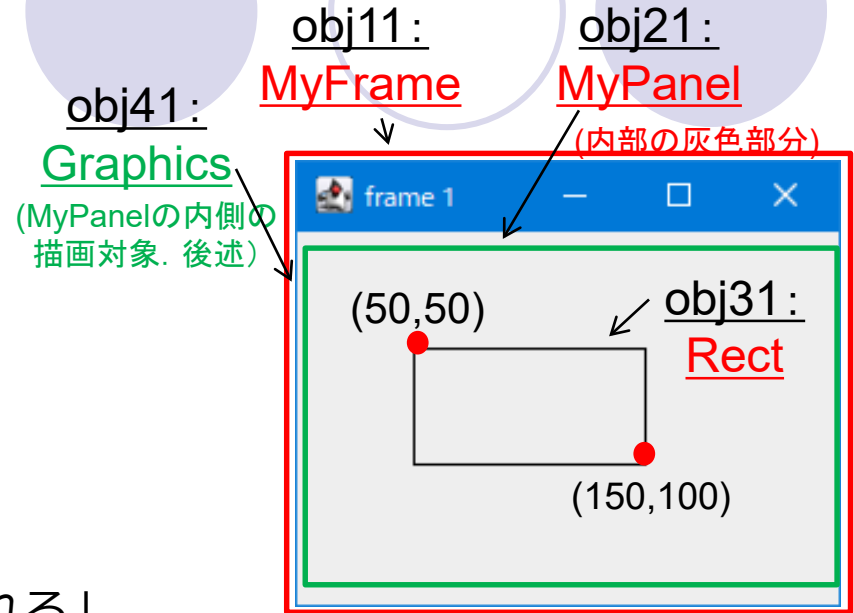
- イベントへの応答を定義することが, オブジェクト指向なGUIプログラミングの中心
- イベント駆動 (ドリブン) 型 ⇔ フロー駆動型
- イベントリスナー (イベントハンドラ)
 - 特定のイベントの種類にどのように反応するかを定義したクラス. そのインスタンスの特定メソッドが呼び出される.
 - あとの回で詳しく説明する
- 今回は「再描画」イベントへの反応を練習しよう

今回の目標

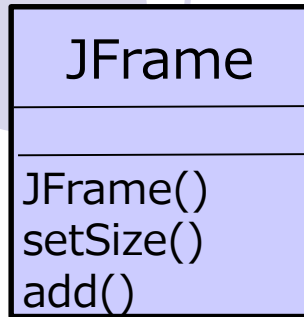
- Windowを作成・表示し，四角形を描画する
 - (1) Windowを作成・表示する (OOP4-A)
 - (2) Windowに四角形(Rect)を「登録」する
 - Rect クラスのインスタンスを生成して，Windowのインスタンスに「登録」する.
 - (3) 描画する
 - イベントを受ける (paintComponent()が呼ばれる)
 - Window に登録されている Rect クラスのインスタンスを描画する (そのインスタンスのdrawメソッドを呼び出す)
 - Rectクラスに自分を描画するメソッド draw を実装する.
- 今回は1つの四角形を描画する
 - 今回は1つのWindow に1つの四角形だけの場合
 - Rectクラスの draw()

画面とクラス/インスタンスの対応

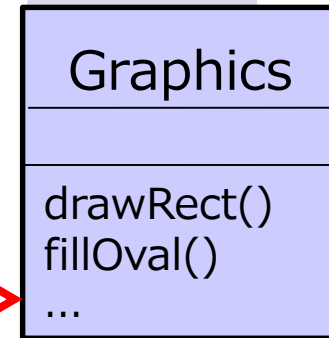
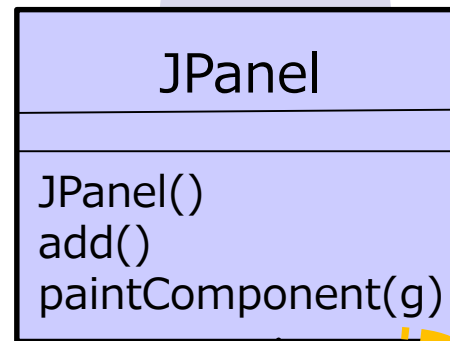
- Window: (ウィンドウの枠)
 - MyFrameクラスのインスタンス
 - Swing の JFrame クラスを拡張して、定義する
- Panel (ウィンドウ内の描画領域)
 - MyPanelクラスのインスタンス.
 - Swing の JPanel クラスを拡張して、定義する
 - MyFrame インスタンスに「組み込まれる」.
- 四角形 (描画される四角形の枠)
 - 先週のRectクラスを拡張する.
 - MyPanel インスタンスに「登録」される.
 - 登録されたMyPanel インスタンスの内側に「描画される」
 - 今回は1つだけ.
- 描画対象 (Panel の内側)
 - Swing (AWT) の Graphics クラスのインスタンス. 基本的な描画機能を持つ.



クラス図



Java API
(Swing)

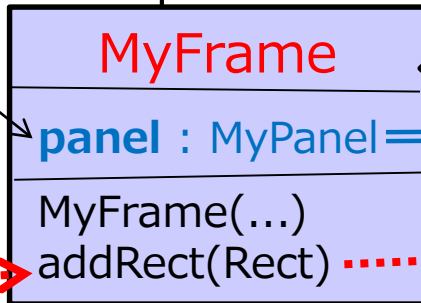


作成する
クラス

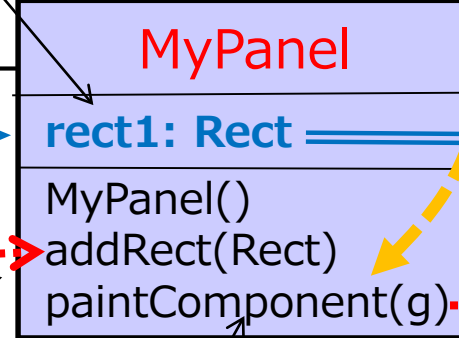
自分に含まれるRectインスタ
ンスを保持するフィールド

メソッド
オーバーライド

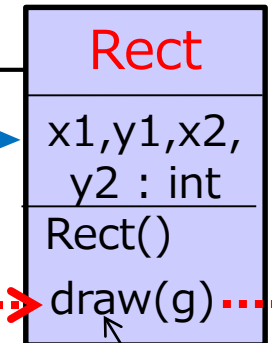
自分の内側の
MyPanel
インスタンス
を保持する
フィールド



Rectインスタンスを
MyPanel インスタンスに
登録する




MyPanel 内を描画する
(JPanelの同名メソッドを
オーバーライド)
Rectインスタンスの
`draw()`を呼び出す。



自分自身（四角形）を
描画する。
Graphics インスタンス
のメソッドを
呼び出す。

メソッドの全体の流れ

- 
- (1) MyFrame/MyPanelインスタンスの生成
 - main メソッドから MyFrame のコンストラクタを呼ぶ.
 - (2) Rectインスタンスの生成と登録
 - main メソッドから, Rectクラスのコンストラクタを呼んで, インスタンスを生成する.
 - addRect() メソッドでMyFrame 経由で MyPanel インスタンスに登録する. 今回はRect型の **rect1** フィールドに代入.
 - (3) 描画 (イベント駆動)
 - Java システムから「再描画イベント」の発生として通知される. MyPanelインスタンスの paintComponent() メソッドが自動的に呼び出される.
 - そのMyPanelインスタンスの **rect1** フィールドに登録されている, Rectインスタンスの draw() を呼ぶ.

JFrame クラス (javax.swing.JFrame)

- 基本的なウィンドウを表す
 - <https://docs.oracle.com/javase/jp/8/docs/api/javax/swing/JFrame.html>
 - java.awt.Window, Container, Component の下位クラス
 - 他の Component (部品) を内部に含むことができる。
- 例題で使用するメソッド (覚える必要はない)
 - JFrame(String title) ※コンストラクタ
 - 指定されたタイトルで、新しい JFrame を作成する。
 - void setLocation(int x, int y) ※Window から継承
 - 左上座標を (x, y) にセットする。
 - void setSize(int width, int height) ※Windowから継承
 - サイズを width および height に変更する。
 - void setVisible(boolean b) ※Windowから継承
 - Window を表示または非表示にする。
 - Component **add**(Component comp, **LAYOUT**) ※継承
 - comp をFrame内部に**組み込む**。 **LAYOUT**は配置の指定

MyFrame のコンストラクタ

- Swing で定義されている JFrame クラスのコンストラクタやメソッドを使って、生成して、位置などを設定する

```
public class MyFrame extends JFrame {
    public MyFrame(String title, int x, int y, int w, int h) {
        super(title);
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.setLocation(x, y);
        this.setSize(w, h);
        this.setVisible(false);
    }

    public void makeVisible() {
        this.setVisible(true);
    }
}
```

MyFrameクラスの
コンストラクタ

引数: タイトル文字列, 左上座標(x,y), 幅w, 高さh

JFrameクラスのコンストラクタの呼び出し

後述

JFrameの上位クラスで定義済な
継承メソッドの呼び出し

JFrameの生成時には非表示状態

MyFrame インスタンスを
表示状態にするメソッド

表示データが揃った後で
表示状態に変更する。

JPanel クラス (javax.swing.JPanel)

- 基本的なコンテナを表す
 - <https://docs.oracle.com/javase/jp/8/docs/api/javax/swing/JPanel.html>
 - javax.swing.Jcomponent, awt.Container の下位クラス
 - 他の Component (部品) を内部に含むことができる.
 - = コンテナ (awt.Container) クラス
- 例題で使用するメソッド
 - JPanel() ※コンストラクタ
 - 新しい JPanel を作成する.

MyPanel のコンストラクタと作成

- Swing の JPanel を拡張して, MyPanel クラスを定義する.
- JPanelクラスのコンストラクタをそのまま使って, 生成する.

```
public class MyPanel extends JPanel {  
    public MyPanel() {  
        super();  
    }  
}
```

MyPanelクラスの
コンストラクタ

JPanelクラスの
コンストラクタ
の呼び出し

- 生成した MyPanel インスタンスをMyFrameに組み込む.

```
public class MyFrame extends JFrame {  
    private MyPanel panel = null;  
    public MyFrame(String title) {  
        ...  
        this.panel = new MyPanel();  
        this.add(this.panel, BorderLayout.CENTER);  
    }  
}
```

自分の内側のMyPanel
インスタンスを保持する
フィールド. null で初期化

MyPanel インスタンスを
生成して, 自分のフィー
ルド panel に登録

上位クラスから継承されているadd()を呼び出して,
「自分」に, MyPanel インスタンスを組み込む

後述

クラス型変数のリテラル値 : null

- オブジェクトへの参照先が「ない」ことを表す
 - クラス型の変数・フィールドの値はオブジェクトへの「参照」．特定のインスタンスを指し示している．
 - null は参照先が「ない」，なにも指していない，ことを表す．
 - クラス型フィールドの初期値だが，明示的な初期化を推奨
 - ローカル変数は初期化しないとコンパイルエラー
 - 値が null な状態で，クラス型変数を用いてはいけない．
 - メソッドの呼び出しやフィールドへのアクセス
 - 実行時に，例外 “NullPointerException” が生成され，止まる．
 - 必ず値が null ではないことをチェックしてから，使う．
 - null チェックと慣例的に呼ばれている．

```
Rect r1 = null;
r1.move(10,10); ×
if (r1 != null) r1.move(10,10);
r1 = new Rect(50,50,80,80)
if (r1 != null) r1.move(20,20);
```

Rect型変数 r1 を
null で初期化

変数 r1 の値が null
な状態でメソッドを
呼び出しているので，
例外が発生する。¹⁵

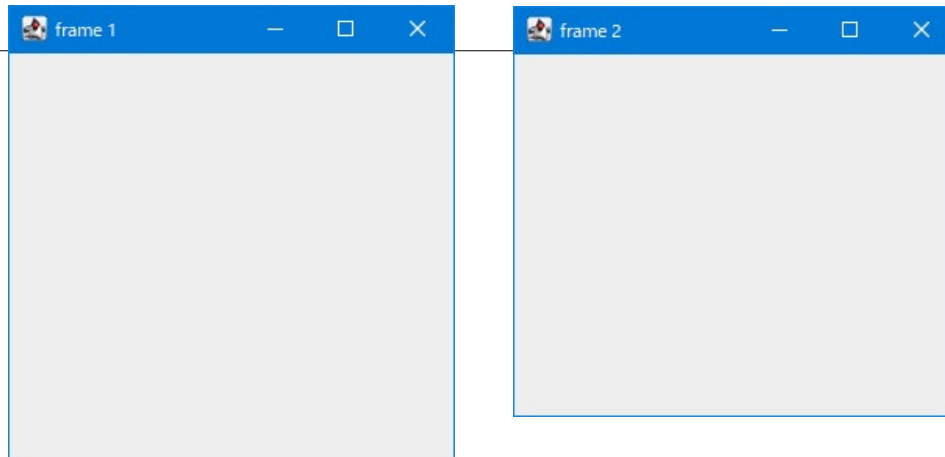
MyFrame/MyPanelの生成(OOP3-A)

- Main クラスの main() メソッド内で, MyFrame クラスのコンストラクタを呼び出す.

```
public class Main {  
    public static void main(String[] args) {  
        MyFrame.setUI(); ← UIを WindowsとMacで統一するための  
                           メソッド (次々スライド)  
  
        MyFrame mf1 = new MyFrame("frame 1", 50, 50, 300, 300);  
        MyFrame mf2 = new MyFrame("frame 2", 70, 70, 300, 270);  
        mf1.makeVisible();  
        mf2.makeVisible();  
    }  
}
```

MyFrameインスタンスの生成

表示状態にする



MyFrame/MyPanelの生成(OOP4-A)

- MyFrameクラス

```
public class MyFrame extends JFrame { MyFrameクラスの  
    private MyPanel panel = null; コンストラクタ  
  
    public MyFrame(String title, int x, int y, int w, int h) {  
        super(title);  
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);  
        this.setLocation(x, y);  
        this.setSize(w, h);  
        this.panel = new MyPanel();  
        this.add(this.panel, BorderLayout.CENTER); ※すべて大文  
        this.setVisible(false); 部品の配置レイアウトの 字な変数は  
    } 指定 (BorderLayoutクラ 「定数」を  
    (次スライドへ続く) スの定数 CENTER) 表す
```

ここまでのプログラム(OOP4-A)

● MyFrameクラス (続き)

```
// (class method)
// set UI as cross-platform Look and Feel
public static void setUI() {
    try {
        UIManager.setLookAndFeel(UIManager.getCross
            PlatformLookAndFeelClassName());
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

実際は一行

UIを WindowsとMacで
統一するための処理.

クラスメソッドであることや、
例外処理(try-catch)を使っている
ことも含めて、今は、理解できなくて
よいです。

ここまでのプログラム(OOP4-A)



- MyPanelクラス

```
public class MyPanel extends JPanel {  
  
    public MyPanel() {  
        super();  
    }  
  
    public void addRect(Rect r) {  
        // insert here  
        this.repaint();  
    }  
  
    @Override  
    public void paintComponent(Graphics g) {  
        ...  
    }  
}
```

次の(2)登録で説明します

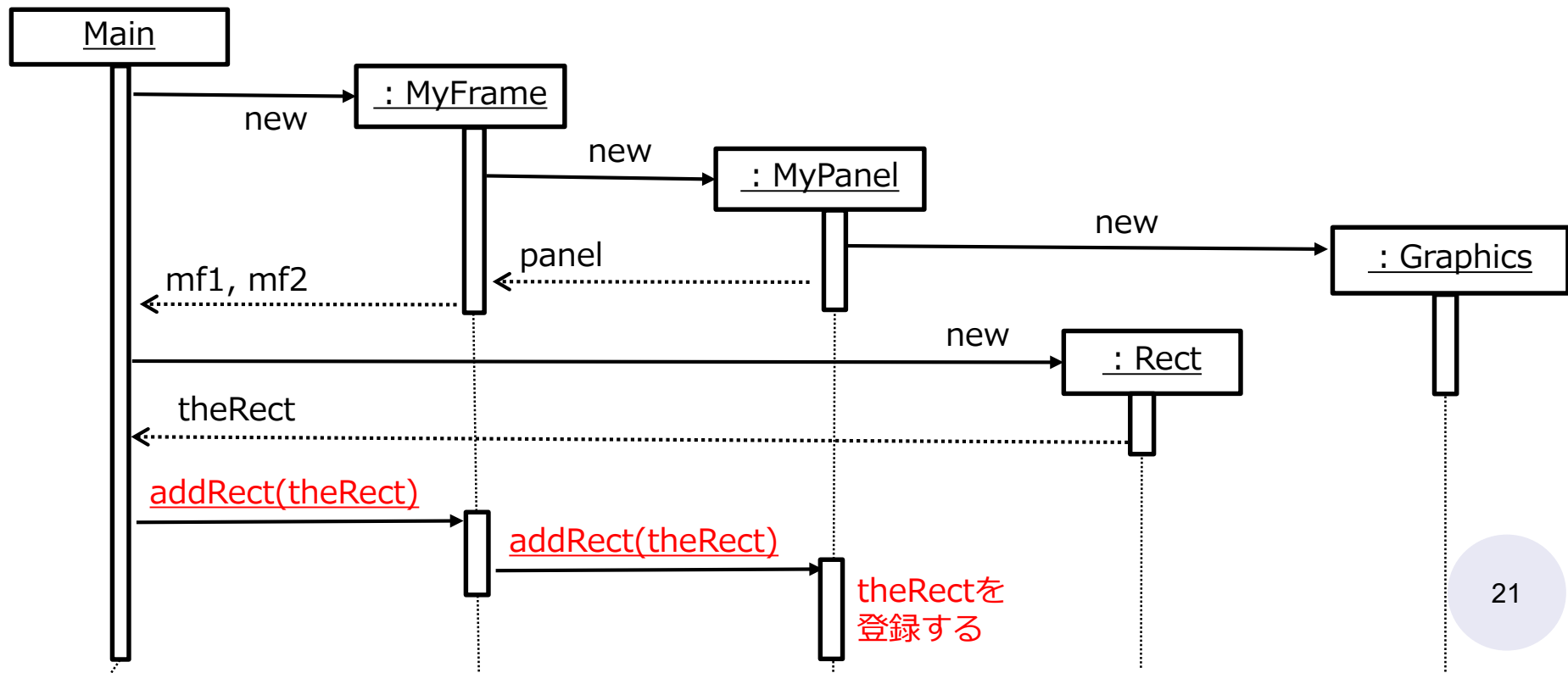
(3)描画で説明します

メソッドの全体の流れ：(2)登録

- (1) MyFrame/MyPanelインスタンスの生成
 - main メソッドから MyFrame のコンストラクタを呼ぶ.
-  (2) Rectインスタンスの生成と登録
 - main メソッドから, Rectクラスのコンストラクタを呼んで, インスタンスを生成する.
-  addRect() メソッドでMyFrame 経由で MyPanel インスタンスに登録する. 今回はRect型の **rect1** フィールドに代入.
- (3) 描画 (イベント駆動)
 - Java システムから「再描画イベント」の発生として通知される. MyPanelインスタンスの paintComponent() メソッドが自動的に呼び出される.
 - そのMyPanelインスタンスの **rect1** フィールドに登録されている, Rectインスタンスの draw() を呼ぶ.

(2)登録のメッセージの流れ

- Mainクラスの main() : MyFrame のインスタンスに向けて, addRect(theRect) を呼ぶ.
- MyFrame のaddRect() : 自分の内側のMyPanelインスタンス (**panelフィールド**) に向けて, addRect(theRect)を呼ぶ.
- MyPanel のaddRect() : 自分の**フィールド**に theRect を登録する.
 - MyPanelの **rect1 フィールド**に代入する.



Rect の作成と登録(1) : main

- MyFrameとRectのインスタンスを作成する.
- そのMyFrameのaddRect()を呼ぶ.
 - void addRect(Rect)

```
public class Main {  
    public static void main(String[] args) {  
        MyFrame.setUI();  
        MyFrame mf1 = new MyFrame("frame 1", 50, 50, 300, 300);  
        MyFrame mf2 = new MyFrame("frame 2", 70, 70, 300, 270);  
        Rect r1 = new Rect(50, 50, 150, 100);  
        mf1.addRect(r1); ← MyFrameクラスのインスタンス mf1 の  
        mf1.makeVisible(); ← addRect() メソッドを呼んで、  
        ← 表示データが揃った後で Rectクラスのインスタンス r1 を登録する。  
        mf2.addRect(new Rect(70, 50, 130, 140))  
        mf2.makeVisible(); ← Rectクラスのインスタンスを生成し、  
        mf2 の addRect() メソッドを呼んで、登録する。  
    }  
}
```

Rect の作成と登録(2) : MyFrame

- MyFrameクラスの addRectメソッド
 - **panel** フィールド が自分の内側のMyPanelインスタンス
 - それに向けて addRect(r)を転送するだけ

```
public class MyFrame extends JFrame {  
    private MyPanel panel = null;
```

```
    public MyFrame(String title) {  
        super(title);
```

```
        ...
```

```
        this.panel = new MyPanel();
```

```
        ...
```

```
    }
```

MyFrameクラスの
addRect()メソッド

```
    public void addRect(Rect r) {  
        if (this.panel != null) {  
            this.panel.addRect(r);  
        }  
    }
```

MyPanelクラスの
addRect()メソッドを
呼ぶ。

Rect の作成と登録(3) : MyPanel


- addRect(Rect)メソッド
 - 今回は自分のRectクラス型フィールド `rect1` に代入する。
 - これで MyPanel に Rect のインスタンスが「登録」された。
 - 図形が増えたので、自分の再描画をリクエストする。
 - 自分の状態に変化が生じたときには、`this.repaint()` を呼ぶ。
 - 後述する `paintComponent()` が自動的に呼ばれる。

```
public class MyPanel extends JPanel {  
    private _____ = null;  
  
    public MyPanel() {  
        super();  
    }  
  
    public void addRect(Rect r) {  
        _____;  
        this.repaint();  
    }  
}
```

Rectクラス型のインスタ
ンスフィールド`rect1`を宣
言し、`null`で初期化する。

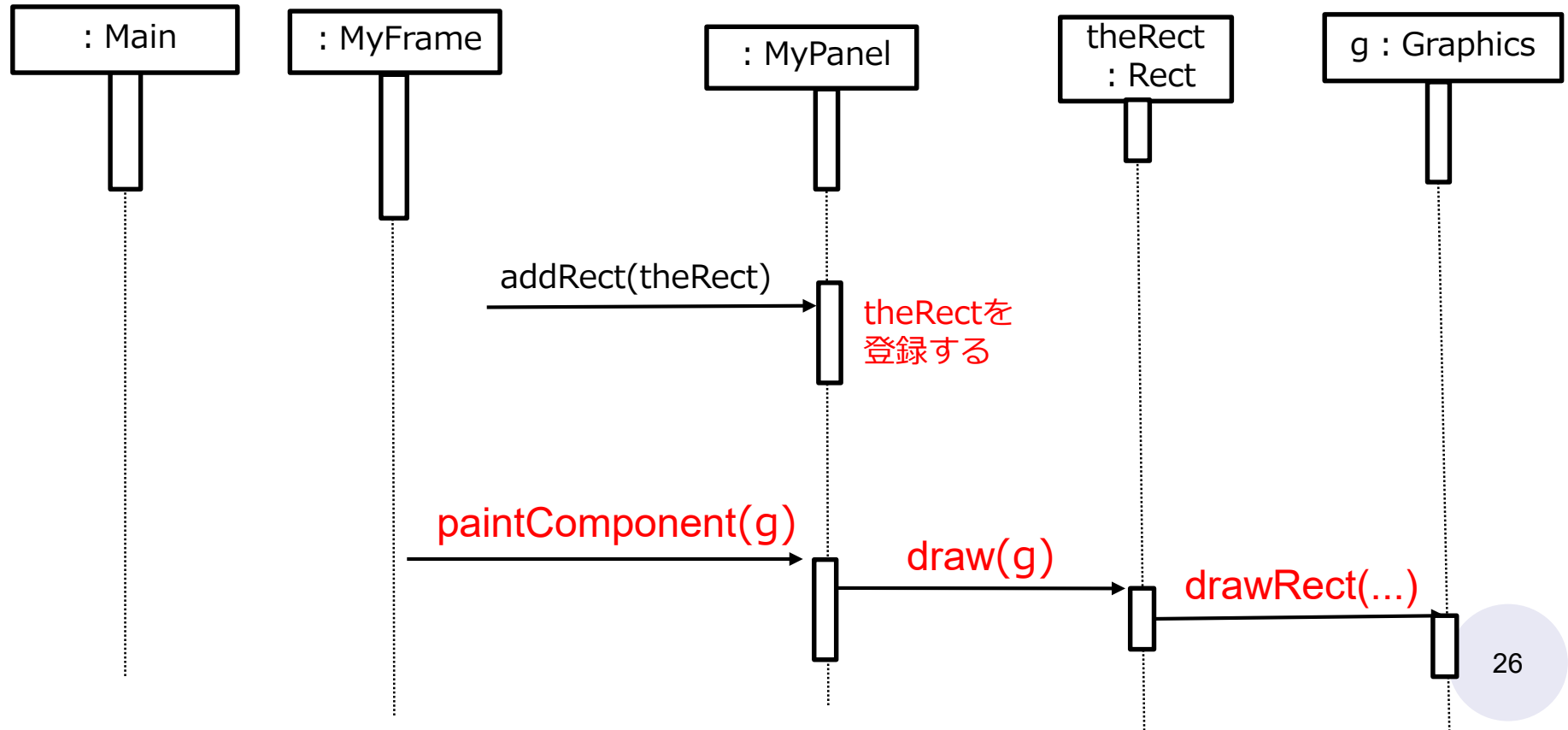
インスタンスフィールド
`rect1`に仮引数の`r`を代入
する

メソッドの全体の流れ：(3)描画

- (1) MyFrame/MyPanelインスタンスの生成
 - main メソッドから MyFrame のコンストラクタを呼ぶ.
- (2) Rectインスタンスの生成と登録
 - main メソッドから, Rectクラスのコンストラクタを呼んで, インスタンスを生成する.
 - addRect() メソッドでMyFrame 経由で MyPanel インスタンスに登録する. 今回はRect型の **rect1** フィールドに代入.
-  (3) 描画 (イベント駆動)
 - Java システムから「再描画イベント」の発生として通知される. MyPanelインスタンスの paintComponent() メソッドが自動的に呼び出される.
 - そのMyPanelインスタンスの **rect1** フィールドに登録されている, Rectインスタンスの draw() を呼ぶ.

(3)描画のメッセージの流れ

- MyPanel の paintComponent(Graphics g)
 - 登録されている Rect インスタンスの draw(g) を呼ぶ.
- Rect の draw(Graphics g)
 - 自分自身を, Graphics g に描画する.



イベント駆動による描画

- Swing では再描画が必要になると `paintComponent` メソッドが自動的に呼ばれる
 - 例：ウィンドウが移動した, リサイズされたなど
 - 引数は Graphics クラスのインスタンス (次スライド)
 - JPanel クラスの `paintComponent` メソッドを自分のクラスで「オーバーライド」して, 自分が描画したい内容を記述する
 - いつ、誰が再描画を起こす (トリガーする) のかは暗黙的
 - 今回は, 実際には `MyFrame` が `visible` になったときに描画される.
 - 例えばコンストラクタで一度描画するだけでは, 不十分.

```
public class MyPanel extends JPanel {
```

再描画の必要が生じると, このメソッドが呼ばれる

@Override

```
public void paintComponent (Graphics g) {
```

← 描画対象 (次ページ)

```
    super.paintComponent (g); ← まず JPanel の描画をする
```

```
    System.out.println("MyPanel painting ...");
```

```
    // insert here ← ここに MyPanel 独自の描画を記述する
```

```
}
```

Graphics クラス (java.awt.)

- 描画対象を表す

- <https://docs.oracle.com/javase/jp/8/docs/api/java/awt/Graphics.html>
- 現在の描画コンテキストを（フィールドとして）持つ
 - 例：現在の色

- 基本的な描画メソッドを持つ

- `void drawLine(int x1, int y1, int x2, int y2)`
 - 点 (x1, y1) と点 (x2, y2)との間に現在の色を使って線を描く.
- `void drawRect(int x, int y, int width, int height)`
 - 指定された矩形の輪郭を描く. 左上x座標, 左上y座標, 幅, 高さ
- `void fillOval(int x, int y, int width, int height)`
 - 指定された矩形の中の楕円形を現在の色で塗りつぶす.
- `void setColor(Color c)`
 - 現在の色を、指定された色に設定する.

描画(1): MyPanel

- MyPanel クラスの paintComponent メソッド
 - フィールド `rect1` に登録されている Rect インスタンスへ向けて, `draw` メソッドを, `g` を引数として, 呼び出す.

```
public class MyPanel extends JPanel {  
    private _____ = null; ← Rectインスタンスへの参照を  
保持するフィールド  
  
    public void addRect(Rect r) {  
        _____; ← 値の代入  
        this.repaint();  
    }  
  
    @Override rect1 に登録されている四角形インスタンスへ  
描画メソッド draw を呼び出す.  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        System.out.println("MyPanel painting ...");  
        if ( _____ != null) {  
            _____;  
        }  
    }  
}
```

描画(2) : Rectクラス

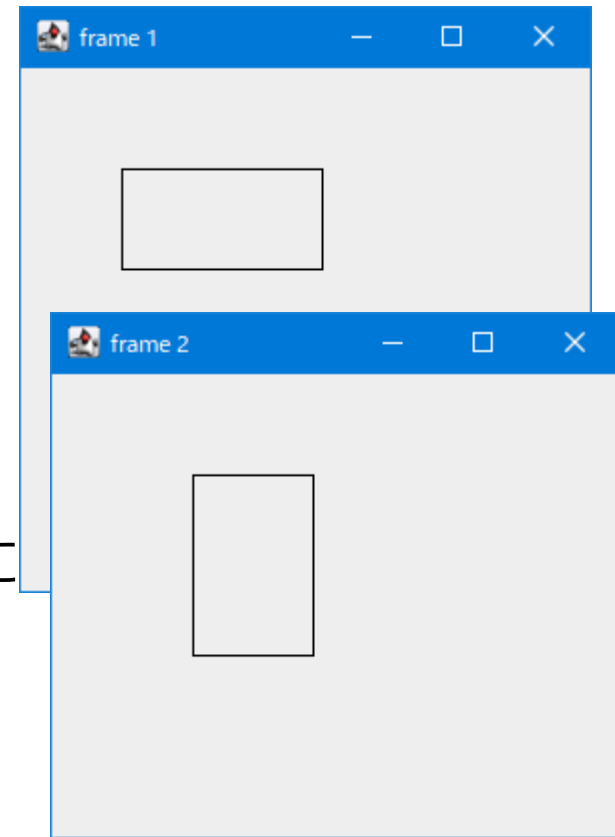
- Rect クラスの draw メソッド

- 自分のフィールドやメソッドを引数にして, Graphicsインスタンス **g** の drawRect() メソッドを呼び出すことで, 自分を描画

```
public class Rect {  
    private int x1, y1, x2, y2;  
    ← 左上座標(x1,y1), 右下座標(x2,y2)  
    public int getWidth() {return (this.x2-this.x1);}   
    public int getHeight() {return (this.y2-this.y1);}   
  
    public void draw(Graphics g) {  
        System.out.println("Rect drawing = " +  
            this.toString());  
        g.setColor(Color.BLACK); ← 描画色を黒にセットする  
        _____ (_____) ;  
        } ← g へ drawRect(左上x座標, 左上y座標, 幅, 高さ)を呼び出す。
```

今回の講義のまとめ

- オブジェクト指向プログラミング(3) : GUI
 - オブジェクト指向の典型的なプログラミングスタイル
 - (1) APIのクラスの「拡張」
 - (2) イベントに反応するメソッド（イベント駆動型）
 - Java API, Swing
 - ウィンドウ
 - イベント
 - 描画（paintComponentイベント）
 - 例題：図形の描画プログラム
 - Window に四角形を描画する
 - 最初のステップとして、1つのWindowに1つずつの四角形
 - 来週、複数の他の図形も描画できるように拡張する。



mainメソッドの内容 :

Rect の作成と登録(1) : main

- FrameとRectのインスタンスを作成する.
- そのFrameのaddRect()を呼ぶ.
- 初期設定のみ. あとはイベント駆動.

```
public class Main {  
    public static void main(String[] args) {  
        MyFrame.setUI();  
        MyFrame mf1 = new MyFrame("frame 1", 50, 50, 250, 200);  
        MyFrame mf2 = new MyFrame("frame 2", 70, 70, 250, 250);  
        Rect r1 = new Rect(50, 50, 150, 100);  
        mf1.addRect(r1);  
  
        mf2.addRect(new Rect(70, 30, 130, 130));  
    }  
}
```

← MyFrameクラスのインスタンス mf1 の addRect() メソッドを呼んで, Rectクラスのインスタンス r1 を登録する.

← Rectクラスのインスタンスを生成し, mf2 の addRect() メソッドを呼んで, 登録する.

イベント駆動による描画

- Swing では再描画が必要になると `paintComponent` メソッドが自動的に呼ばれる
 - 例：ウィンドウが移動した, リサイズされたなど
 - 引数は `Graphics` クラスのインスタンス
 - `JPanel` クラスの `paintComponent` メソッドを自分のクラスで「オーバーライド」して, 自分が描画したい内容を記述する
 - いつ、誰が再描画を起こす（トリガーする）のかは暗黙的
 - 今回は, 実際には `MyFrame` が `visible` になったときに描画される.
 - 例えばコンストラクタで一度描画するだけでは, 不十分.

```
public class MyPanel extends JPanel {
```

再描画の必要が生じると, このメソッドが呼ばれる

@Override

描画対象 (次ページ)

```
public void paintComponent(Graphics g) {
    super.paintComponent(g); ← まず JPanel の描画をする
    System.out.println("MyPanel painting ...");
    // insert here ← ここに MyPanel 独自の描画を記述する
}
```