# プログラミング環境特論

## JavaによるＧＵＩプログラミングと JavaBeans入門

筑波大学
佐藤



## JavaBeans

- JavaBeansとは、BuilderとよばれるＧＵＩツールで(vitsual)プログラミングができるようにするための標準のインタフェースを実装したJavaのクラスである。
- このJavaBeansの仕様にしたがってソフトウエアの部品をつくっておくことによって、これらを適宜組み合わせることによって、プログラミングすることができるようになり、再利用可能なソフトウエア部品を開発する環境を提供する



## JavaBeansでは何ができるのか

- Builderのデザインシートの上に、お絵描きソフトの感覚でクラスを張り付けることができる。
- 変数値（プロパティ）をプロパティシートに羅列された個々のプロパティエディタ上で変更し、保存できる。
- ２つのクラスに対して特定の「イベントオブジェクト」の交換を指定することで連携させることができる。
- デザインシートごと保存して、実行可能なJavaプログラムとすることができる。



## JavaBeansによる開発

- Beansライブラリを組み合わせてJavaプログラムを開発する。 すでに、JavaBeansとして開発されたボタンやアイコンなどを組み合わせて簡単にプログラムを作ることができる。
- オリジナルのBeansライブラリを開発する。必要な部品は自分で開発して、これを組み込んだり、他の人に提供したりすることができる。



## JavaによるGUIプログラミング

- Javaの開発環境であるJDK(Java Development Kit)には、ＧＵＩをもつプログラムの作成を容易にするために、Abstract Window Toolkit(AWT)というパッケージが提供されている。AWTは、ボタンやメニューなどのＧＵＩ部品、イメージの表示、マウスやキーボードのからの入力（イベント）を処理する機能を提供している。これらの部品は特定のwindowシステムに依存しないようになっている。最近では、ＡＷＴに変わってSwingというライブラリが使われているようである。
  - AWTが提供しているクラスは、以下のものである。
  - UI部品：ボタンやメニューなどの部品、それらを配置する入れ物に当たるパネルやウインドウクラス。
  - 配置管理：部品をどのように配置するかを管理するためのLayoutクラス
  - その他：グラフィックスライブラリ。



## 例

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.util,* ;
class HelloGui extends Applet implements ActionListener {
   Button button1,button2;
   Label label;
   public void init(){
      setLayout(new BorderLayout());
      setLocale(Locale.JAPANESE);
      button1 = new Button("hello");
      button2 = new Button ("clear");
      button1.addActionListener(this);
      button2.addActionListener(this);
      label = new Label("");
      add(label,"North");
      add(button1, "Center");
      add(button2,"East");
   }
    public void actionPerformed(ActionEvent e){
```

プログラミング環境特論

## 例

```
public void actionPerformed(ActionEvent e){
        if(e.getSource() == button1) label.setText("hello");
          if(e.getSource() == button2) label.setText("");
}
public static void main(String args[]){
        HelloGui h  =  new HelloGui();
        Frame f  =  new Frame("hello GUI");
          hello.init();
          hallo.start();
          f.add(hello,"Center");
          f.setSize(300,100);
          f.show();
      }
}
```

プログラミング環境特論

## アプレットとJavaアプリケーション

- このプログラムは、Applet をextendsしており、アプレットとしても使うことができるようになっている。ついでに、アプレットとJavaアプリケーションの違いを述べておこう。
- アプレットとは、webページに組み込まれるJavaプログラムである。webページに
  <APPLET CODE="HelloGUI.calss" WIDTH=300 HEIGHT=100>
  </APPLET>
  と書き、このページと同じところにおいておけば、webブラウザがこのjavaプログラムをネットワークを使ってダウンロードし実行してくれる。
- アプレットにするためには、java.applet.*をインポートし、appletをextendsして作成する。アプレットクラスはContinerクラスのサブクラスであるPenelクラスをサブクラスとなっており、ブラウザにおいて、このアプレットクラスを生成し、ロードされたアプレット（つまり、HelloGUI）ブラウザのウインドウ（これがFrameにあたる）を表示してくれる。したがって、アプレットとして実行する場合にはFrameは必要ない。

プログラミング環境特論

## コンポーネントのレイアウト

- レイアウトのための設定する。
  setLayout(new BorderLayout());
  このBorderLayoutは、コンテナを５つの領域（上、下、右、左、中央）にわけてそのどこかに配置するレイアウトである。これをコンテナに設定するメソッドがsetLayoutで、部品を入れるときには、addを使う。入れる部品について、インスタンス変数を宣言しておき、
  Button button1;  // 部品のインスタンスの変数を宣言
- 次に、initにおいて、インスタンスを生成し、
  button1 = new Button("hello!!!");  // 部品を作る
  部品をコンテナにいれる。
  add(button1,"East");
- なお、labelは、文字を表示するための部品である。

プログラミング環境特論

## イベント処理

- 通常のＣの入出力では、getcやreadの処理が呼び出さないと入力は行われない。
- これに対して、JavaのＧＵＩ部品は自律的に動いていると考えることができる。すなわち、マウス操作はキーボード入力を行うと何かの処理が実行される。これをイベントとよび、このイベント処理を記述することがＧＵＩプログラミングの中心的な部分となる。（実際、Javaでは別にスレッドが動いており、それらが入力について監視していると考えることができる）オブジェクト指向言語であるJavaでは、イベント自身もイベント処理もオブジェクト指向のフレームワークで構成されている。

プログラミング環境特論

## イベント処理

- イベントはイベントが発生したところ（event source）から、そのイベントをうけとって処理をするところ（listerner）に伝えられる。リスナーにはイベント処理のメソッドが定義されており、イベントに応じた処理をすることになる。これは、代理人リスナーモデル(delegation-based Listener Model)と呼ばれ、Java1.1で導入されたモデルである。
- イベントを受け取って、処理をするリスナーの機能を持たせるためにはjava.util.EventListenerインタフェースである<EventType>Listenerインタフェースをインプリメントする。たとえば、ボタンをクリックした時に生成されるActionEventを受け取るのはActionEventListenerインタフェースを実装したクラスで、このリスナークラスのactionPerformedメソッドにボタンを押したときに起こる振る舞いを記述する。

プログラミング環境特論

## イベント処理

- イベントハンドラの設定
  - button1.addActionListener(this)
- button1に関するイベントが起きたときに、リスナーである、このクラスのインスタンスに伝えられactionPerformedが呼び出されるようになる。ここで引数になっているのが伝えられたイベントactionEventである。このactionEventにはどこから伝えられたイベントなのかという情報が含まれており、actionPerformedではこれをつかって、labelにある文字を消したり、表示したりしている。
  if(e.getSource() == botton1) label.setText("hello");
- イベントには、マウスが移動したり、スクロールバーが移動したりといった様々なイベントがあり、リスナーがある。このプログラムでは、リスナーをこのクラスのオブジェクト自身が受け取っているが、別に設定することも可能である

プログラミング環境特論

## JavaのReflection機能

- Reflectionとは、「反映」「反射」という意味であるが、プログラムが（他の）プログラムを調べるという意味
  - java.lang.reflect
  - あるクラス（クラスファイル）にどのようなフィールド（インスタンス変数）、メソッド、Constructorがあるかを調べる
  - オブジェクトのフィールドの値を読み書きしたり、メソッドを適当な変数を与えて呼び出すことができる。

プログラミング環境特論

## 例

- ab.java

```
public class ab {
  public int a;
  public int b;
  public ab(int x, int y){
    a = x;
    b = y;
  }
  public ab() { a = 1; b = 1; }
  public int getA() { return a; }
  public void setA(int x) { a = x; }
  public int plus() { return a + b; }
  public void print() {
    System.out.println("a="+a+",b="+b);
  }
}
```

プログラミング環境特論

## 例

- report.java フィールド名を調べる

```
import java.lang.reflect.*;

public class report {
  public static void main(String argv[]){
    Class cls;
    try {
      cls = Class.forName("ab");
    } catch(Exception e){
      System.out.println("cannot instantiate class");
      return;
    }
    try {
      System.out.println("Field:");
      Field fields[] = cls.getDeclaredFields();
      for(int i = 0; i < fields.length; i++){
       System.out.println(" "+fields[i].toString());
      }
    } catch(Exception e){
```

プログラミング環境特論

## フィールド情報の取得

- クラスの情報を取得するのがClass.forNameである。

```
Class cls;
cls = Class.forName("ab");
```

- フィールド、すなわちインスタンス変数の情報を取得するメソッドが、getDeclaredFields
  - Fieldsオブジェクトの配列を返す。
  - Fieldsオブジェクトはclsにあるフィールドの情報が入る。
  - これをtoStingメソッドで文字列に変換して、出力する。

プログラミング環境特論

## 例

- test.java 値をセットしてみる

```
import java.lang.reflect.*;
public class test {
  public static void main(String argv[]){
    Class cls;
    Object obj;
    try {
      cls = Class.forName("ab");
    } catch(Exception e){
      System.out.println("cannot instantiate class:"+e);
      return;}
    try {
      Field a = cls.getField("a");
      Field b = cls.getField("b");
      Class pTypes1[] = { Integer.TYPE, Integer.TYPE };
      Constructor Cons =cls.getConstructor(pTypes1);
      Class pTypes2[] = { Integer.TYPE };
```

プログラミング環境特論

## フィールドへの値のセット

- 名前を指定して、フィールド情報を取るのがgetField

```
Field a = cls.getField("a");
```

- constructorの情報を取得する関数が、getConstructor
  - このメソッド関数では引数のタイプ情報を与える。
  - タイプ情報は、Classの配列で、primitiveタイプの場合にはデータ型のラップクラス（primitiveをクラスオブジェクトとして扱うためのクラス）に定義されている。

```
Class pType1[] = {Integer.TYPE,Integer.TYPE};
Constructor Cons=cls.getConstructor(pType1);
```

## 例

- test.java 値をセットしてみる

```
Class pTypes2[] = { Integer.TYPE };
Method setA = cls.getMethod("setA",pTypes2);
Class pTypes3[] = { };
Method getA = cls.getMethod("getA",pTypes3);
Method plus = cls.getMethod("plus",pTypes3);
Method print = cls.getMethod("print",pTypes3);
Object VOID[] = { };
Object args[] = { new Integer(20), new Integer(1) };
obj = Cons.newInstance(args);
System.out.println("a="+a.getInt(obj));
System.out.println("b="+b.getInt(obj));
a.setInt(obj,10);
print.invoke(obj,VOID);
System.out.println("a="+a.getInt(obj));
System.out.println("b="+b.getInt(obj));
Object o = plus.invoke(obj,VOID);
```



## フィールドへのアクセス

- メソッド情報を取得する関数がgetMethodで、こちらはメソッド名と引数の情報を与える。

```
Method setA = cls.getMethod("setA", pType2);
```

- Constructorクラスの newInstanceメソッドを呼び出す。引数については、すべて、オブジェクトの配列として与える。したがって、primitiveタイプの場合は、ラップクラスを使う。

```
object args[] = { new Integer(10), new Integer(20)};
object obj = Cons.newInstance(args);
```



- フィールド情報を使って、フィールドを読み出す場合には、getIntをつかう。ここでは、フィールドを「intとして」読み出すことと注意。

```
a.getInt(obj);
```

- メソッドを呼び出すときには、invokeメソッドを使う。

```
setA.invoke(obj,arg);
```



## JavaBeans

- JavaBeans
  1. Builderのデザインシートの上に、お絵描きソフトの感覚でクラスを張り付けることができる。つまり、Builderというツールが（外部の）Beansを操作することができる。
  2. 属性（プロパティ）をプロパティシートに羅列された個々のプロパティエディタ上で変更し、保存できる。つまり、ロードしたBeansの状態を変更することができる。
  3. 2つのクラスに対して特定の「イベントオブジェクト」の交換を指定することで連絡させることができる。
  4. デザインシートごと保存して、実行可能なJavaプログラムとすることができる。



## reflection & introspection

- reflectionの機能は、１，２の機能を実現するために使われている。
- Beansは、JavaBeansの仕様で記述されたクラス（ファイル）である。
- 基本的にはどのクラスファイルでもBeansとして扱うことができるが、JavaBeansの仕様で書いておくことによって、Builderで内部の値を変更できるようになる。
- これを行うのがJava Beansのintrospection機能である。これは、reflection機能を使って実装されており、高レベルのreflection機能と言える。



## Beansとは

- JavaBeansは、JavaBeansの仕様に従ったクラスオブジェクト
- JavaBeansにある属性を持たせるには、JavaBeansの仕様に従ったメソッドを定義する。クラスの属性は大体はクラスのフィールド（インスタンス変数）で実現されることが多いが、その属性をアクセスするメソッドの名前で決める。
  - たとえば、あるBeansがforegroundという名前のプロパティも持つのは、そのbeansがColor getForegroud()とvoid setForegroud(Color c)というメソッドを持つ場合である。
- プロパティをセットするメソッドをsetter、取り出すメソッドをgetterという。あるプロパティに対し、
  setter= “set” + プロパティ名
  getter = “get” + プロパティ名

## 例

- propという属性を持つbean

```
public class oneProp {
        private int p = 9;
        public int getProp() { return p; }
        public int setProp(int i) { p = i; }
    }
```

- booleanプロパティの場合には、getterの名前は、”is”から始まる。



## JavaBeans イベント処理

- JavaBeansでもお互いのイベントを交換するために、AWTで解説したEvent Listenerモデルを使っている。
  - これは、イベント発生する側にどこにそのイベントを伝えるかを設定し、イベントが伝えられる側にリスナーメソッドを定義するものである。
- あるJavaBeansがfooという名前のイベントを発生するとする。このとき、beansはイベントセットfooを持つという。
  - このbeansはfooEventというイベントオブジェクトの送り手とならなくてはならない。すなわち、このbeansにこのイベントの受け手であるFooListenerを登録することができるメソッドaddFooListenerがあることを意味する。



## イベント処理

- javaBeansでは、プロパティと同様に規則的な名前をつけることによって、定義する。
  - Eventオブジェクト
    - class “イベント”Event extends java.util.EventObject
  - Listener
    - interface “イベント”Listener extends java.util.EventListener
  - Listerner登録
    - void add”イベント”Listener(“イベント”Listener listener)
  - Listerner抹消
    - void remove”イベント”Listener(“イベント”Listener listener)
- JavaBeansでは、プロパティと同様に、このような名前の規則に従ったメソッドを探すことによって、イベントセットを見つける。



## 例

- 一つのイベントセットfooをもつbeans

```
import java.util.*;
public interface FooListener extends { … }

import java.beans.*;
public class eventDesc {
  public void addFooListener(FooListener l){ … }
  public void removeFooListener(FooListener l) { … }
}
```



## introspection

- Builderでは以上にみるようなプロパティやイベントセット、メソッドを持っているかを調べて、それらを操作する。
- このような性質を調べる機能がintrospectionである
- intropsectorクラスを用いることによって調べることができるようになっている。

```
import java.beans.intropsector;
….
 Beans Info = Intropsector.getBeanInfo(Class
beanClass);
```

- reflectionを利用して実装されている。



## 例

- test.java

```
import java.lang.reflect.*;
import java.beans.*;

public class test {
  public static void main(String argv[]){
    Class cls;
    Object obj;
    try {
      cls = Class.forName("ExplicitButton"/*"OurButton"*/
    } catch(Exception e){
      System.out.println("cannot instantiate class:"+e);
      return;
    }
    try {
      BeanInfo info = Introspector.getBeanInfo(cls);
      System.out.println("info = " + info);
```

## 例

- test.java

```
try {
      BeanInfo info = Introspector.getBeanInfo(cls);
      System.out.println("info = " + info);
      PropertyDescriptor[] pds = info.getPropertyDescript
      System.out.println("properties:");
      for(int i = 0; i < pds.length; i++){
        System.out.println(" "+ pds[i].getName());
      }
      EventSetDescriptor[] esd = info.getEventSetDescript
      System.out.println("EventSet:");
      for(int i = 0; i < esd.length; i++){
        System.out.println(" "+ esd[i].getName());
      }
      MethodDescriptor[] mds = info.getMethodDescriptors(
      System.out.println("Method:");
      for(int i = 0; i < mds.length; i++){
```



## Serialization

- オブジェクトの状態をファイルやネットワーク上に書き出したり、読み込み復元したりする機能である。
- これを使って、状態をセーブする
- Serializableインタフェースが必要

```
Date d = new Date(); ….
FileOutputStream fout = new FileOutputStream(“tmp”);
ObjectOutputStream out = new ObjectOutputStream(fout);
out.write(d);
out.flush();
```

このファイルから、呼び出してオブジェクトを復元するには、

```
FileInputStream fin = new FileInputStream(“tmp”);
ObjectInputStream in = new ObjectInputStream(in);
Date d = (Date)in.readObject();
```



## Wiringとは

- JavaBeansでのプログラミングの中心となるのが、Wiring、すなわちそれぞれのJavaBeansのイベントを通じての関連づけをする部分である
  - Builder内で、部品Aと部品Bについて、マウスをつかって結びつける操作を行う。
  - すなわち、この操作では部品Aで発生したイベントをBにつたえて、部品Bのメソッドを呼び出すようにする。



## イベント処理についての復習

- ボタンbuttonAがactionEvent というイベントセットをもち、それをHelloLabelがうけとって、”hello”というメッセージを表示する場合、おこなわなくてはならないことは、HelloLabelをActionListenerとしてインタフェースをつくっておき、これをbuttonAのリスナーとして登録すること

```
class buttonA { …
  buttonA(HelloLabel label) {
     … addActionListener(label); …
  } }
class HelloLabel implements ActionListener { …
   actionPerformed(ActionEvent ev) {
 …     showHello(); /*“hello”を表示*/ }
}
class Exec {
  public static void main(String argv[]) {
    HelloLabel l = new HelloLabel();
    new buttonA(l):
}
```



## 問題点

- Execは初期設定をするためのmainを持つクラスである。
- しかし、このようなプログラムをするには、プログラムの中にイベント処理のためのコードが埋め込まれてしまっている。
- このようなコードが埋め込まれていては、「部品」として他の用途につかうことができないということになってしまう。



## Adapterクラスの利用

- イベントを受け取るためのAdapterという仲介をするオブジェクトをつくって、イベントの橋渡しをさせることによって、元のコードにリスナーを作らなくてもよくなる。

```
class buttonA { … }
class HelloLabel { … }
class Adapter implements ActionListener {
  private HelloLabel target;
  public void setTarget(HelloLabel t){ target = t; }
  public void actionPerformed(ActionEvent ev) {
            target.showHello(); }
}
class Exec {
   public static void main(String argv[]){
     Hello l = new HelloLabel();
     buttonA botton = new buttonA();
     Adapter adapter = new Adapter();
     adapter.setTarget(l):
     button.addActionListener(adapter);
}
```

プログラミング環境特論

## BuilderでのWiringの処理

- 簡単なBuilderであるBeanBoxでは、Wiringに対して以下の処理を行っている。
  - Adapterクラスのプログラムを BeanBoxの中で実行中に生成し、コンパイルして、それを動的にロードして実行している。
    - __Hookup_????という名前のクラスのプログラムを作り、これがadaptorになっている。これをコンパイルし、できたクラスファイルを動的にロードする。
  - BeanBoxでadaptorにtargetを設定したり、adaptorをListenerとして設定したりするにはreflection機能を利用してクラスのメソッドを呼び出している。
    - 、java.lang.reflectのMethodクラスに定義されているinvokeメソッドをつかって、setTargetやaddActionListenerを呼び出せばよい。

プログラミング環境特論

## Appletの生成

- BeanBoxでは適当に部品を配置し、wiringしてできたプログラムをAppletとしてsaveすることができる。
  - MyApplet.html: appletをテストするためのHTMLファイル
  - MyApplet_files: ここには生成されたappletのプログラムとdataが入る。
  - MyApplet.jar: MyApplet_filesをJARにしてまとめたもの。
  - その他の必要なbeansの入ったJARファイル

```
<applet
    archive="./myApplet.jar,./support.jar
        ,./mytools.jar
    "
    code="Hello"
    width=390
    height=492
>
```

プログラミング環境特論

## Jarファイルのつくり方

- javaのクラスのファイルは、jarというコマンドでJARファイルにしておくことができる。
- 実際のクラスのあるdirectoryに対するパスを設定する代わりに、このファイルをクラスパスに設定しておけばこの中にあるクラスファイルが参照されるようになる。
- jarファイルの作り方は、tarに似ている。クラスのあるdirectoryで、

  % jar –cvf directory

  とすればよい。
- 実際、jarのファイル形式は、zip形式を使ったものなので、zip/unzipコマンドでも作ることができる。

プログラミング環境特論

## Beansの作り方

- 自分が作ったBeanをBeanBoxなどのBuilerにもっていくためにはJARファイルにしておかなくてはならない。但し、単なるJARファイルではなくて、

  Name: クラスファイル名
  Java-Bean: true
  Name: クラスファイル名
  Java-Bean: true
  …

  というどれがBeanであるかというファイルを作って、これをmanifestとして作らなくてはならない、
- このファイルをmanifest.tmpとすると、

  %jar cfm JARFILE manifest.tp *.class

  として作る。
- mはmanifestファイルを指定するオプションである。これをクラスパスに設定しておけば、builderで使えるようになる。

プログラミング環境特論

## Java GUIのプログラミング環境のまとめ

- AWT
- Javaのイベントモデル - event-listenerモデル
- applet
- reflection
- serialization
- Beansの考え方