プログラミング言語論 #11

Java

2018-07-02

3 番目のオブジェクト指向プログラミング言語として Java を紹介する

特徴 1：グローバル変数がない
特徴 2：分割コンパイルへの対応
特徴 3：インターフェイス
特徴 4：型と例外

Java

特徴

拡張子	`java`
コンパイル	`javac`
実行	`java`（インタプリタ）
型に関する特徴	強い静的型付け, OOPL
	`interface` による関係拡張（仕様継承）
	実行速度改善のため全てをクラスにしなかった
その他の特徴	設計方針： *Compile Once, Run Everywhere*

Java/1

Main.java

```
1 public class Main {
2   public static void main (String[] argv) {
3     System.out.println("Hello");  // (A)
4   }
5 }
```

- main の返値の型は void で，引数は文字列の配列を受け取る．これは C の main にとても似ている． (A): System パッケージの中の標準出力オブジェクト out に，定義されている println というメソッドを呼び出す（関数適用している）．
- 配列型の書き方は少し違う（String argv[] ではない）．
- if 文， while 文，コメントほぼ同じ， int 型， float/double 型, char 型ある．

Java/2

Main.java?

```
1  int x = 0;
2
3  void main (String[] argv) {
4       System.out.println("Hello");
5  }
6
7  public class Main {
8    public static void main (String[] argv) {
9       System.out.println("Hello");
10   }
11 }
```

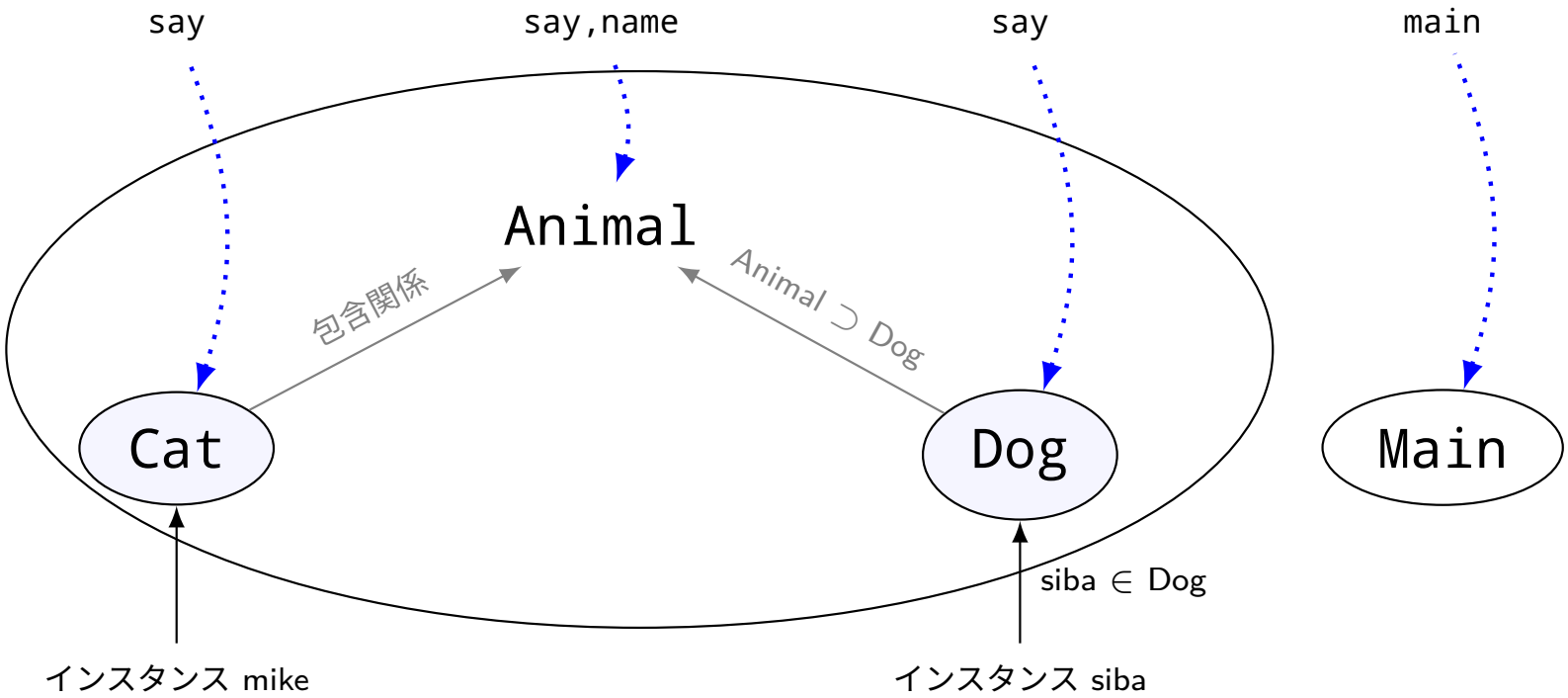
クラス以外のグローバルな識別子の定義は許されない

Java/3

a class and its sub classes

```
1  class Animal {
2    public String name = "";  // グローバルなインスタンス変数
3    public void say () { System.out.println("..."); } // メソッド
4  }
5  class Cat extends Animal {
6    public void say () { System.out.println("nya"); }
7  }
8  class Dog extends Animal {
9    Dog (String a) { name = a; }   // コンストラクタ
10   public void say () { System.out.println(name + ": one!"); }
11 }
12 public class Main {
13   public static void main (String[] argv) {
14      Cat mike = new Cat();         // 未定義？
15      Dog siba = new Dog("siba");  // インスタンス生成
16      mike.say();   // これはC++と同じ呼び出し方
17      siba.say();
18   }
19 }
```

このファイルの中のクラス関係



- Cat という集合（クラス）は Animal という集合（クラス）に包含される（部分集合）; Cat のインスタンスは自動的に Animal のインスタンスともみなせる
- Dog という集合（クラス）は Animal という集合（クラス）に包含される（部分集合）；Dog のインスタンスは自動的に Animal のインスタンスともみなせる

Java でのファイル分割

ファイル名と public なクラス名の対応

Animal.Java

```
1  public class Animal {
2    public String name = "";  // グローバルなインスタンス変数
3    void say () { System.out.println("..."); } // メソッド
4  }
```

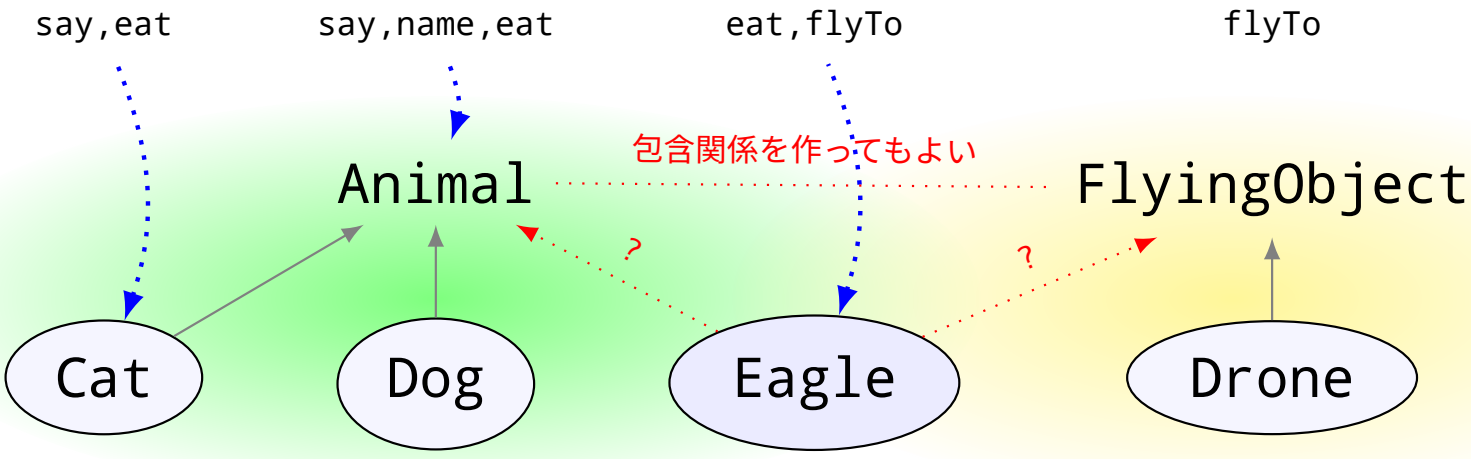
Main.java

```
1  class Cat extends Animal {
2    void say () { System.out.println("nya"); }
3  }
4
5  class Dog extends Animal {
6    Dog (String a) { name = a; }   // コンストラクタ
7    void say () { System.out.println(name + ": one!"); }
8  }
9
10 public class Main {
11   public static void main (String[] argv) {
12      Cat mike = new Cat();         // 未定義？
13      Dog siba = new Dog("siba");  // インスタンス生成
14      mike.say();   // これはC++と同じ呼び出し方
15      siba.say();
16   }
17 }
```

この延長にパッケージ： package, import

単純な包含関係の限界

multiple inheritance



Eagle は eat メソッドも flyTo メソッドもあるべき

- Animal ⊃ Eagle では， Eagle が飛べない
- FlyingObject ⊃ Eagle では， Eagle が食事しない
- Animal ⊃ Eagle & FlyingObject ⊃ Animal では， Dog も飛べる
- FlyingObject ⊃ Eagle & Animal ⊃ FlyingObject では， Drone も食事する

全ておかしい： Java ではクラス間の関係を 2 種類使うことで解決

extends と implements

class + implements

```
1  class Animal {      // こちらは従来通りクラスとして実装
2    void eat () { ... }
3  }
4  interface Fly {    // 飛べるということの抽象的なグループ
5   void flyTo (...); // Flyに属するために必要なメソッドの宣言
6  }
7
8  class Eagle extends Animal implements Fly {
9    void eat() {...}    // 定義は必要ではない
10   public void flyTo(...) { ... }  // Flyの実装が必要
11 }
12 class Dog extends Animal ...
13 class Drone implements Fly ...
14 // class Drone extends Robot implements Fly  ...
```

- Eagle, Animal は extends によって Animal に包含
- Eagle, Drone は implements によって「飛べるもの」に所属

try catch 構文

GetPage.java

```
10      try {
11          writer.println(message);// ネットワーク障害の可能性
12          writer.flush();
13      } catch (Exception ex) { ex.printStackTrace(); }
```

```
17      try {
18          while((message = reader.readLine()) != null) {
19          System.out.println(message);
20          }
21      } catch (Exception ex) { ex.printStackTrace(); }
```

例外はエラーを一般化した概念：例

- 通信できない
- 0 での割り算
- 対象ファイルの書き込み権限がない

C の場合

例外処理はない．ただし多くの標準関数（システムコール）はエラーが起きたかどうかを int で返す

```
1  int e;
2  e = printf("画面表示");
3  if (e == -1) {      //  画面表示できなかった場合の処理
4  }
5  e = receive(&message); // ネットに接続されていない
6  if (e == -1) {      //  受信できなかった場合の処理
```

欠点：

- 各行で検査が必要（1 対 1 対応）なのでプログラミングが面倒
- 検査は強制されないので，エラーに対応しないプログラムがコンパイルに通ってしまう

Exception handling; 例外処理

Java での例外処理構文

```
1 try 例外を起こすかもしれない文（ブロック）
2  catch (対応したい例外の種類1) その時に実行する文（ブロック）
3  catch (対応したい例外の種類2) その時に実行する文（ブロック）
4  catch ...
```

例外は言うなれば特殊な返値なので関数の型の一部とみなせる (thrown)．従って，呼び出し側は例外が起きた時の処理を書かないとコンパイル時に「型が違う」エラーになる．

Python での例外処理構文

```
1 try: 例外を起こすかもしれない文（ブロック）
2 except 対応したい例外の種類1: その時に実行する文（ブロック）
3 except 対応したい例外の種類2: その時に実行する文（ブロック）
4 ...
5 else: 例外未発生の時に実行する文  # このelseは省略可能
```

構文はある．動的型付け言語なのでコンパイル時チェックはない（強制されない）．

closing

- OOPL とは何か
- インスタンスとは何か
- 関数とメソッドの違いを説明せよ
- コンストラクタとは何か
- 多くのプログラミング言語に現れる public とは何か
- 講義では使わなかったが「継承」とは何か
- 講義では使わなかったが「オーバーロード」とは何か
- OOPL を一つ選びその言語で ad-hoc 多相の機能を持つ証明（コンパイルエラーにならない実例）を示せ
- OOPL を一つ選びその言語で parametric 多相の機能を持つ証明（コンパイルエラーにならない実例）を示せ