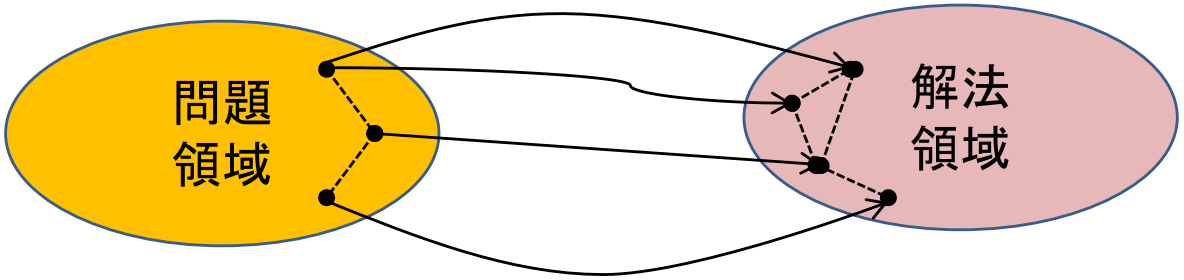
プログラミングD
Java講義 第2回：
オブジェクト指向の基本

肥後 芳樹

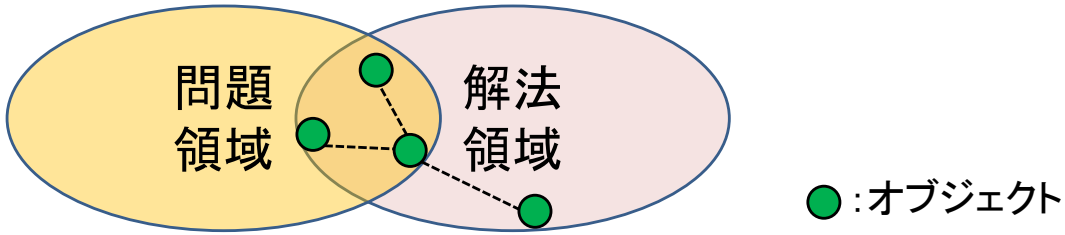
プログラミング言語と抽象化

- プログラミング言語：抽象化を与えるもの
  - アセンブリ言語：機械語の抽象化だが依然抽象度が低い
- 命令型（imperative）プログラミング言語
  - 実行すべき命令列で構成，手続き型プログラミング言語
    - 例：FORTRAN，BASIC，C言語など
  - アセンブリ言語の抽象化を与えるが，コンピュータの構造を意識したプログラミングが必要
  - 問題領域と解法領域間の関連付けが必要
    - このマッピングや保守（再利用を含む）にコストがかかる



プログラミング言語と抽象化

- オブジェクト指向プログラミング言語（Object-oriented Programming Language: OOPL）
  - オブジェクト（Object）：問題領域に存在する要素の解法領域における表現法
    - 問題領域上の概念をオブジェクトとして表現することで，問題領域の概念を用いてコーディング可能
      - コンピュータ上の概念もオブジェクトで表現
    - オブジェクトは小さいコンピュータのようなもの
      - 状態（State）と操作（Operation）を持つ



オブジェクト指向言語の5つの特徴

- Alan KayによるSmalltalkの特徴の要約
  - Smalltalk：最初に成功したOOPL

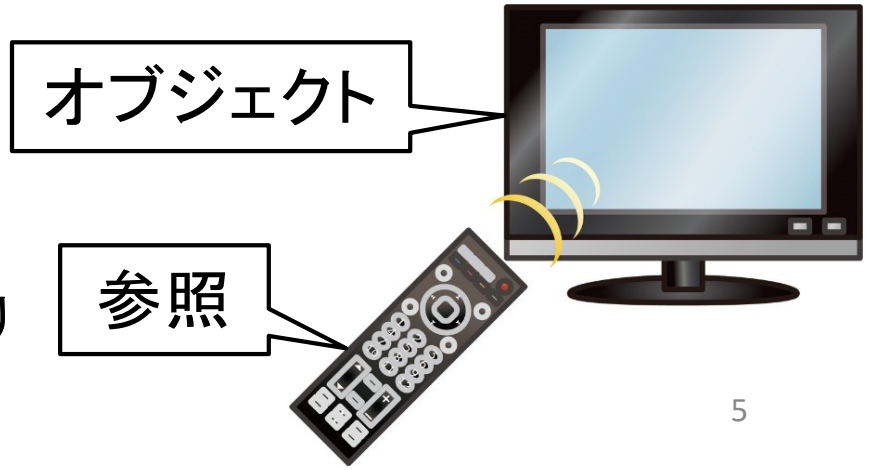
  - 全てのものがオブジェクト
  - オブジェクト同士はメッセージにより通信
  - オブジェクトは他オブジェクトにより構築可能
  - すべてのオブジェクトは型（Type）を持つ
  - 同一の型に基づくオブジェクトは同一メッセージを受信可能

オブジェクトへの参照

- Javaでは（ほぼ）すべてのものがオブジェクト
  - プログラミング言語にはメモリ上の要素に対するアクセス手段が必要（ex. C, C++：ポインタ）

  →Javaではオブジェクトへの参照（Reference）を提供

- 参照に近い概念：テレビのリモコン
  - リモコン（参照）を通じてテレビ（オブジェクト）を操作可能
  - テレビが無くても存在可能
    - String s;
  - テレビとは別物
    - オブジェクトと関連付ける必要あり



オブジェクトの生成

- 参照を作ったら，オブジェクトと関連付ける
  - 参照の初期値はnull
  - null：オブジェクトを指していないことを意味する

- オブジェクトの作成：通常は new 演算子 を利用

```
String s = new String(“tube”);
```

  - 新しいString（文字列）型のオブジェクトを作成
    - Stringの場合は，String s = “tube”;とも記述可能

  - Stringのような既成の型以外の型も構築可能

5種類のデータ保存場所

- レジスタ（Registers）：もっとも高速だが限りのある格納領域
  - （C，C++と異なり）Javaでは直接指定できない
- スタック（Stack）：RAM上に存在
  - 生存期間に厳密であり，Javaでは参照，変数などを格納
- ヒープ（heap）：RAM上に存在
  - 生存期間が緩く，Javaではオブジェクトを格納
  - new演算子により，領域が確保
  - スタックよりも確保・開放に時間がかかる
- メソッドエリア：変更されない定数，プログラムの格納場所
- 非RAM領域：プログラムとは独立したデータの保存場所
  - 永続オブジェクト（Persistent object）

プリミティブ型は例外

- Javaでのオブジェクト作成
  - newでヒープ上に領域を確保し，スタック上の参照と関連付ける
- Javaでのオブジェクトアクセス
  - 小型で頻繁に使うデータには効率が悪い

→プリミティブ型（Primitive Types）：小型で頻繁に使うデータのために用意されている型

- newを使わない
- 参照ではなく，直接値を保持
- スタック上に格納される

Javaのプリミティブ型

プリミティブ型	サイズ	最小値	最大値	Wrapper
boolean	—	—	—	Boolean
char	16bits	Unicode 0	Unicode 2^{16}-1	Character
byte	8bits	-128	+127	Byte
short	16bits	-2^{15}	$+2^{15}$-1	Short
int	32bits	-2^{31}	$+2^{31}$-1	Integer
long	64bits	-2^{63}	$+2^{63}$-1	Long
float	32bits	IEEE754	IEEE754	Float
double	64bits	IEEE754	IEEE754	Double
void	—	—	—	Void

Wrapper：プリミティブ型と非プリミティブ型オブジェクトとの連携に利用

スコープ

- C，C++と同様，Javaの変数にもスコープがある

```
{
    int x  = 12;
    // Only x available
    {
        int q = 96;
        // Both x & q available
    }
    // Only x available;
    // q is “out of scope”
}
```

```
{
    int x  = 12;
    {
        int x = 96; // illegal
    }
 }
```

これはエラーが発生
（Duplicate local variable）

`//`：以降，行末までコメントアウト

オブジェクトを破棄する必要は無い

```
{
    String s = new String(“a string”);
} // End of scope
```

- 参照sはスコープ外では無効
  - ただし，Stringオブジェクトはメモリを占有したまま
    - C++ではdestroyの必要あり

→Javaはガベージコレクタ（garbage collector）を搭載

- 不要になった（=参照されていない）オブジェクトのメモリ領域を適宜開放
  - メモリ開放忘れによるメモリリークを防止
  - この処理をガベージコレクション（Garbage Collection：GC）と呼ぶ

クラスとオブジェクト

- すべてのものがオブジェクトであるとすると，オブジェクトの型とは？

- クラス（class）：オブジェクトの振る舞い，特徴を定義する型（type）
  - 同じ特徴（データ要素）と振る舞いを持つ型
  - クラスはプログラマが自由に定義できる（built-inのクラスもある）

- オブジェクト：クラスに基いて生成されるもの
  - 各オブジェクトは個別の状態を持つ

クラスの定義

- では，クラスはどのように作るのか？

→classキーワードによって定義

```
class ClassName { /* Class body goes here */ }
```

- 定義された型に対してnewキーワードでオブジェクトを生成可能

```
ClassName a = new ClassName();
```

フィールドとメソッド

- クラス内には2種類の要素を定義できる
- フィールド（field）：性質や状態を表現するもの
  - メンバ変数，属性とも呼ばれる
  - データを格納する領域
    - 通常，オブジェクトごとに独立して値を管理する
  - オブジェクトとプリミティブ型が該当
    - オブジェクトへの参照は，初期化する必要がある
- メソッド（method）：振る舞いを表現するもの（後述）

記述例

- フィールドの記述例

```
class DataOnly{
    int i;
    double d;
    Boolean b;
}
```

- オブジェクト生成

```
DataOnly data
    = new DataOnly();
```

- フィールドの参照方法

```
objectReference.member
```

  - <参照名>.<フィールド名>

- フィールドの参照例

```
data.i = 47;
data.d = 1.1;
data.b = false;
```

- フィールド参照を繰り返すことも可能

```
myPlane.leftTank.capacity
    = 100;
```

プリミティブ型のデフォルト値

プリミティブ型	デフォルト値	プリミティブ型	デフォルト値
boolean	false	int	0
char	‘¥u0000’	long	0L
byte	(byte)0	float	0.0f
short	(short)0	double	0.0d

- このデフォルト値の代入はフィールド（メンバ変数）として利用されるときのみ適用される
  - が，明示的に初期化するのが良い
  - ローカル変数には適用されない

メソッド，引数，返値

- メソッド（method）：振る舞いを表現するもの
  - CやC++の関数のようなもの
  - Javaではオブジェクトが受信するメッセージを決定する
  - 基本形：

```
ReturnType methodName( /* Argument list */ ){
    /* Method body */
}
```

- ReturnType：メソッド呼び出し後に戻ってくる値（返値）の型
- Argument list：メソッドに渡す型と変数名のリスト

※methodNameとArgument listの組はクラス内で一意である必要がある

メソッド，引数，返値

- メソッド呼び出し：

```
objectReference.methodName(arg1, arg2, arg3);
```

- 例：`int x = a.f();`
  - オブジェクトaのメソッドf()を呼び出す
  - 返値とxの型は互換可能でなければならない

- 通常メソッドはオブジェクトに対して呼び出される
  - クラスに対して呼び出されるメソッド（static method）もある

メソッドの引数

- メソッドに送る情報
  - 引数は原則オブジェクト(or プリミティブ型)
  - 定義方法:オブジェクトの型と名前の組のリスト
  - オブジェクト本体ではなく，参照を渡す
  - 例:

```
int storage(String s){
    return s.length * 2;
}
```

  - 値として何も返したくない場合はReturnTypeにvoidを指定
    - その場合，returnは不要

static キーワード

- 通常，オブジェクトをnew演算子で生成し，その後メソッドやフィールド値を利用
- このメカニズムでは都合が悪い場合がある
  - 1つだけ存在すれば良い値を扱う場合
  - オブジェクトが存在していなくてもメソッドが必要な場合

→static：フィールドやメソッドが各オブジェクト固有のものではないことを表すkeyword

  - オブジェクトを生成しなくても，staticメソッドの呼び出しやstaticフィールドへのアクセスが可能
  - staticフィールド／メソッドは，クラスフィールド／クラスメソッドとも呼ばれる

static キーワード（フィールド）

- 使用例

```
class StaticTest {
    static int i = 47;
}

StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();

st1.i++;
StaticTest.i++;
System.out.println(“i: ” + st2.i); ←出力値は？
```

※staticフィールド／staticメソッドはクラス名でアクセスすることが推奨されている

static キーワード（メソッド）

- staticメソッドも同様の利用方法

```
class Incrementable {
    static void increment() { StaticTest.i++}
}

Incrementable sf = new Incrementable();
sf.increment();

Incrementable.increment();
```

- staticメソッドが必要な典型的な例：main()メソッド
  - プログラムの始点であり，オブジェクトが存在しない状態での呼び出される

名前の可視性

- 名前管理
  - 同じ名前のクラス，フィールド，メソッドを利用しても名前衝突が起きないための仕組みが必要
- 名前空間（namespace）の概念を導入
  - Internetドメインを逆順にした名前を利用
  - 例：`org.eclipse.swt.widgets.Button`
    - dotはサブディレクトリを表現
  - 名前空間により，名前の一意性を保証

定義済みクラスの利用

- コンパイラにクラスの場所を提示する必要あり

- import文により，対象クラスを指定できる
  - `import java.util.ArrayList;`
    →コンパイラに，ArrayListを利用することを明示
  - パッケージ内の全てのクラスを参照する場合
    - `import java.util.*;`

最初のJavaプログラム

- 現在の時刻を表示するプログラム

```
// HelloDate.java
import java.util.*;
public class HelloDate {
  public static void main(String[] args) {
    System.out.println("Hello, it’s: " +new Date());
  }
}
```

Import文

- 初めにimport文を記述（import java.util.*;）
  - java.util.Dateクラスは以降，Dateのみでアクセスできる
    - import文を使わない場合，毎回正式名で記述する必要がある
  - ［例外］java.langパッケージ内クラスには，import文は不要
    - 例）Systemクラス，Stringクラスはimport文不要
  - （参考）Javaが標準で提供するクラス
    - https://cr.openjdk.java.net/~iris/se/17/build/latest/api/java.base/module-summary.html
    - クラスや所属パッケージを調べる事が可能
    - 前スライドに登場するクラス，メンバ，メソッドを上記サイトにて確認してみると良い

Javaの基本的な規則

- クラス名とファイル名は基本的に同名
  - ex) クラス名：HelloDate → ファイル名：HelloDate.java
- クラスのうちの1つにmain()を定義する
  - 次のシグネチャを用いること（必須）

```
public static void main(String[] args){
```

  - public：アクセス修飾子の1つ
    - そのクラスの外部からもアクセス可能なことを表す
  - args：引数のリスト（String型のリスト）
    - たとえ引数がなくても上記シグネチャを用いること

コメント

- Javaには3種類のコメントがある
  - /* */:複数行のコメント記述に利用
  - // :その行の終わりまでのコメントアウト
  - /** */:ドキュメント自動生成時に利用(javadoc)

```
/* This is a comment that
   contionues across multiple lines */

// This is a one-line comment

/** HelloDate: our first Java program … */
public class HelloDate{
    …
```

ドキュメントをコメントとして記述

- コードとドキュメントの整合性管理は悩ましい問題
  - Javaでは，コード内コメントの一部を抽出してドキュメントを生成

- Javadoc
  - JDK内ツール群の1つ
  - 出力はHTMLファイル
  - 先に参照したWebサイトもJavadocにより生成されたもの
  - コード変更後の同期的なドキュメント更新が可能

コーディングスタイル

- Java言語の暗黙の記述ルール
  - クラス名：先頭文字を大文字にする
    - 複数のWordをつなげるときはハイフンは使わずに各Wordの先頭文字を大文字にする
  - その他のメソッド，変数，引数，参照名：先頭文字を小文字にする
  - 単語は基本的に省略しない

```
class AllTheColorsOfTheRainbow{
    int anIntegerRepresentingColors;
    void changeTheHueOfTheCOlor(int newHue){
        // …
    }
    // …
}
```