情報処理技法
(Javaプログラミング)2

第8回
継承

人間科学科コミュニケーション専攻
白銀 純子

第8回の内容

- 継承
- オーバーライド
- ポリモーフィズム

前回の出席課題の回答

■「クラス」と「オブジェクト」の違いについて、具体例を交えて説明しなさい。

解答例:
「クラス」とは、プログラム中で扱う様々なもの(現実世界の実物)を分類したものであり、
「オブジェクト」とは、クラスにあてはまる具体的な実物である。具体的には、図書館の
蔵書管理システムを考えると、システムで扱うデータとして、本のデータがある。「本」が「クラス」に
相当する。これは、「本」にあてはまる実物は図書館の中にたくさん存在するためである。また、
「蔵書ID: 0001, 基礎講座 Java, 白銀純子, 毎日コミュニケーションズ,2010」のように、具体的に
ただ1つに特定できる実物がオブジェクトである。

前回の復習

課題のフィードバック～割り算～(1)

- int型での割り算の使い方
  - int型の割り算は、商と余りを計算するテクニックが便利
  - ある数から何かを取り除き、その取り除いた数からまたさらに取り除く、ということを繰り返す場合に有効
    - 引き算ではなく割り算で取り除く場合

課題のフィードバック～割り算～(2)

Ex. 第1回課題のその2: 数を各桁に分解するには?

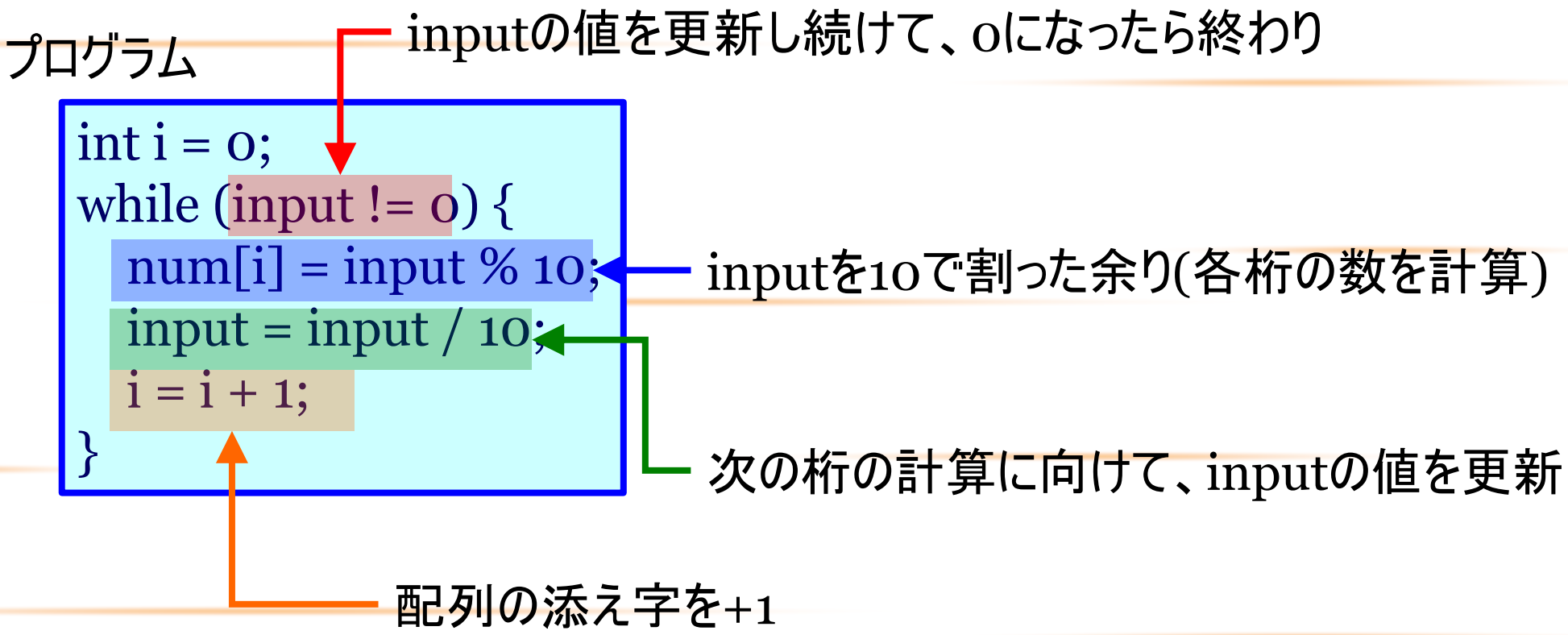
- 分解したい数をinput、各桁を入れる配列をnumとする
- 変数inputの値が123だとすると…

for文で、inputの値を10で割った商と余りを計算する

- iが0のときのinputに対する計算: 1 2 3 →商: 12, 余り: 3
  - ✓ 余りを配列num[0]に入れ、商でinputの値を更新
- iが1のとき: 12 →商: 1, 余り: 2
  - ✓ 余りを配列num[1]に入れ、商でinputの値を更新
- iが2のとき: 1 →商: 0, 余り: 1
  - ✓ 余りを配列num[2]に入れ、商でinputの値を更新
- iが3のとき: 0 →商: 0, 余り: 0 になるので、終了

: 商　　: 余り

課題のフィードバック～割り算～(3)



※標準入力された数を、文字列の段階で数えた文字数がそのまま桁数になるので、
その桁数でfor文で分解してもOK

課題のフィードバック～割り算～(4)

Ex. 第1回課題のその3: 硬貨の枚数を計算するには?

- 枚数を計算したい金額をmoneyとする
- 変数moneyの値が987だとすると…

moneyの値を硬貨の金額で割った商と余りを計算する

- 500円玉の枚数計算: 987 →商: 1, 余り: 487
  - ✓ 商が500円玉の枚数、余りでmoneyの値を更新
- 100円玉の枚数計算: 487 →商: 4, 余り: 87
  - ✓ 商が100円玉の枚数、余りでmoneyの値を更新
- 50円玉の枚数計算: 87 →商: 1, 余り: 37
  - ✓ 商が50円玉の枚数、余りでmoneyの値を更新
- 10円玉の枚数計算: 37 →商: 3, 余り: 7
  - ✓ 商が10円玉の枚数、余りでmoneyの値を更新
- 5円玉の枚数計算: 7 →商: 1, 余り: 2
  - ✓ 商が10円玉の枚数、余りが1円玉の枚数

課題のフィードバック～割り算～(4)

硬貨の金額を設定した配列を用意

各硬貨の枚数を表す配列

```
int i, coin[] = {500, 100, 50, 10, 5, 1}, coinNum = new int[6];
for (i = 0, i < 6; i = i + 1) {
    coinNum[i] = money / coin[i];
    money = money % coin[i];
}
System.out.println("硬貨の枚数は");
for (i = 0; i < 6; i = i + 1) {
    System.out.println(coin[i] + "円玉: " + coinNum[i] + "枚");
}
System.out.println("です。");
```

500円玉から順に
枚数計算

出力も配列で可能

coinNumは、配列0番に500円玉の枚数、1番に100円玉の枚数…というように枚数が設定された配列になる(for文で書くとすっきり書ける)

課題のフィードバック～フラグ～(1)

- ループ文の処理の結果によって、その後の処理内容が変わる場合
  - 単純にif文では処理ができない

「フラグ」という考え方を使う
(「フラグを立てる」という言い方をする)

- 今回の課題で使えそうなところ
  - その1で、暗証番号の入力が成功したかどうか
  - その1で、引き出しの処理が成功したかどうか
  - その4で、目的の数が見つかったかどうか
  - etc.

課題のフィードバック～フラグ～(2)

■ 第1回課題その4

□ 乱数の中に、入力された数が複数存在するかも

実際のプログラムだと…

- 情報検索で、特定のキーワードを含むWebページを探す, etc.
  - ➡ 検索結果が1つだけ、というわけにはいかない

実際に多かったプログラム

```
for(i = 0; i < 100; i = i + 1) {
    if (random[i] == num) {
        i = 101;  (または break;)
    }
}
if (i < 100) {
    System.out.println("数" + num + "は" + i + "番目にありました。");
} else {
    System.out.println("Not Found");
}
```

数が見つかると、ループを強制終了

数が見つかっても、1つだけ出力

課題のフィードバック～フラグ～(3)

- フラグの使い方
  - int型の変数を1つ用意する(ここではflagとする)
    - 変数flagで扱う値が2種類であれば、boolean型でもOK
  - 変数flagに0を代入する
  - ループ文の中で、ある特定の状況になった場合、その状況に応じて1, 2, 3, …という数をflagに代入する
    - ループ文の後の条件分岐の条件に応じて、いくつ分の数があれば良いかを決める
  - ループ文の後でif文でflagの値をもとに条件分岐する

ループ文で特定の状況になったときに旗(フラグ)を立てておくことで、ループの終了後に旗がどこかに立っていれば(いなければ)…ということで処理をする

課題のフィードバック～フラグ～(4)

- Ex. 逐次探索で、探したいものが見つからない場合に「Not found!」と出力する
  - 探したいものが見つかった、ということで旗を立てる

```
int i, flag = 0;
for (i = 0; i < 100; i = i + 1) {
    if (random[i] == num) {
        System.out.println("数" + num + "は" + i + "番目にありました。");
        flag = 1;
    }
}
if (flag == 0) {
    System.out.println("Not found!");
}
```

数が乱数の中に見つかると、
flagの値を変更

flagの値が0のまま
＝ 数が見つからなかった

※この場合、「見つからなかった場合」はflagの初期値としない
(for文で、「見つかった」は判断できるが、「見つからなかった」は判断できないため)

前回の復習

メッセージ

- メッセージ = あるクラスで定義されたメソッドを呼び出すこと
- 「オブジェクト名.メソッド」(または「クラス名.メソッド」)の形式で呼び出し
  - ただし、メソッドを定義している同じクラス内で呼び出すときは、オブジェクト名やクラス名は省略
  - Ex1. str.substring(m, n)
    - Stringクラスに定義されている「substring」というメソッドを呼び出し
      (strはStringクラスのオブジェクト = String型の変数)
  - Ex2. Integer.parseInt(str)
    - Integerクラスに定義されている「parseInt」というメソッドを呼び出し
      (strはStringクラスのオブジェクト = String型の変数)

メソッドの定義

- クラスに関連づけるメソッドとオブジェクトに関連づけるメソッドの2種類
  (内容の定義はこれまでと全く同じ)

クラスに関連づけるメソッドの定義のテンプレート

```
public   static   戻り値のデータ型   メソッド名(引数) {
    メソッドでの処理内容
    return  処理結果;
}
```

オブジェクトに関連づけるメソッドの定義のテンプレート

```
public   戻り値のデータ型   メソッド名(引数) {
    メソッドでの処理内容
    return  処理結果;
}
```

違い:「static」キーワードがついているかいないか

「static」のあるなし(1)

- 「static」がついているメソッド(クラスメソッド)
  - これまで作ってきたメソッド
  - 「クラス名.メソッド」の形式で呼び出し可能(「オブジェクト名.メソッド」の形式でも可能)
  - staticつきのメソッド内部で、同じクラスで定義されているメソッドを呼び出すときは、そのメソッドにもstaticが必要
    - mainメソッドから呼び出すときは、「static」がついている必要
  - メソッドを定義しているクラスのインスタンス変数を利用することは不可能(クラス変数は利用可能)

オブジェクトによって処理結果の変わらないメソッドはstaticをつけて良い
(つけないメソッドにしても良い)

「static」のあるなし(2)

- 「static」がついていないメソッド(インスタンスメソッド)
  - 必ず「オブジェクト名.メソッド」の形式で呼び出し(「クラス名.メソッド」の形式では呼び出し不可能)
  - staticなしのメソッド内部で、同じクラスで定義されているメソッドを呼び出すときは、そのメソッドはstaticはついていても、ついていなくても良い
  - staticなしのメソッド内部で、同じクラスで定義されているフィールドは、インスタンス変数・クラス変数とも利用可能

オブジェクトによって処理結果の変わるメソッド(同じクラスで定義されているstaticなしのフィールドを処理に使うなど)は、必ずstaticなし

mainメソッド

- 「この部分を最初に実行する」という意味のメソッド
  - クラスメソッドの一種
    - 「public static void main(String[] args) {」のメソッド
  - Javaでは、プログラムを実行したときに、まず最初にmainメソッドの「{」と「}」の間に書かれている処理を実行
    - 複数のクラス作成するときは、mainメソッドを作成するのは1つのクラスのみ
    - 複数のクラスを使ってプログラムを実行するときは、「java」コマンドで指定するクラスは、メインメソッドを持っているクラス

オブジェクト同士でのメソッドの呼び出し

- あるクラス(名前: ClassA)で定義されているメソッドを...
  - 別のクラスから呼び出す場合

    メソッドがインスタンスメソッドの場合: ClassAのオブジェクト名.メソッド

    メソッドがクラスメソッドの場合: ClassA.メソッド
  - 同じクラス(ClassA)から呼び出す場合:「オブジェクト名.」や「クラス名.」は不要
    - メソッド名 + 引数のみでOK

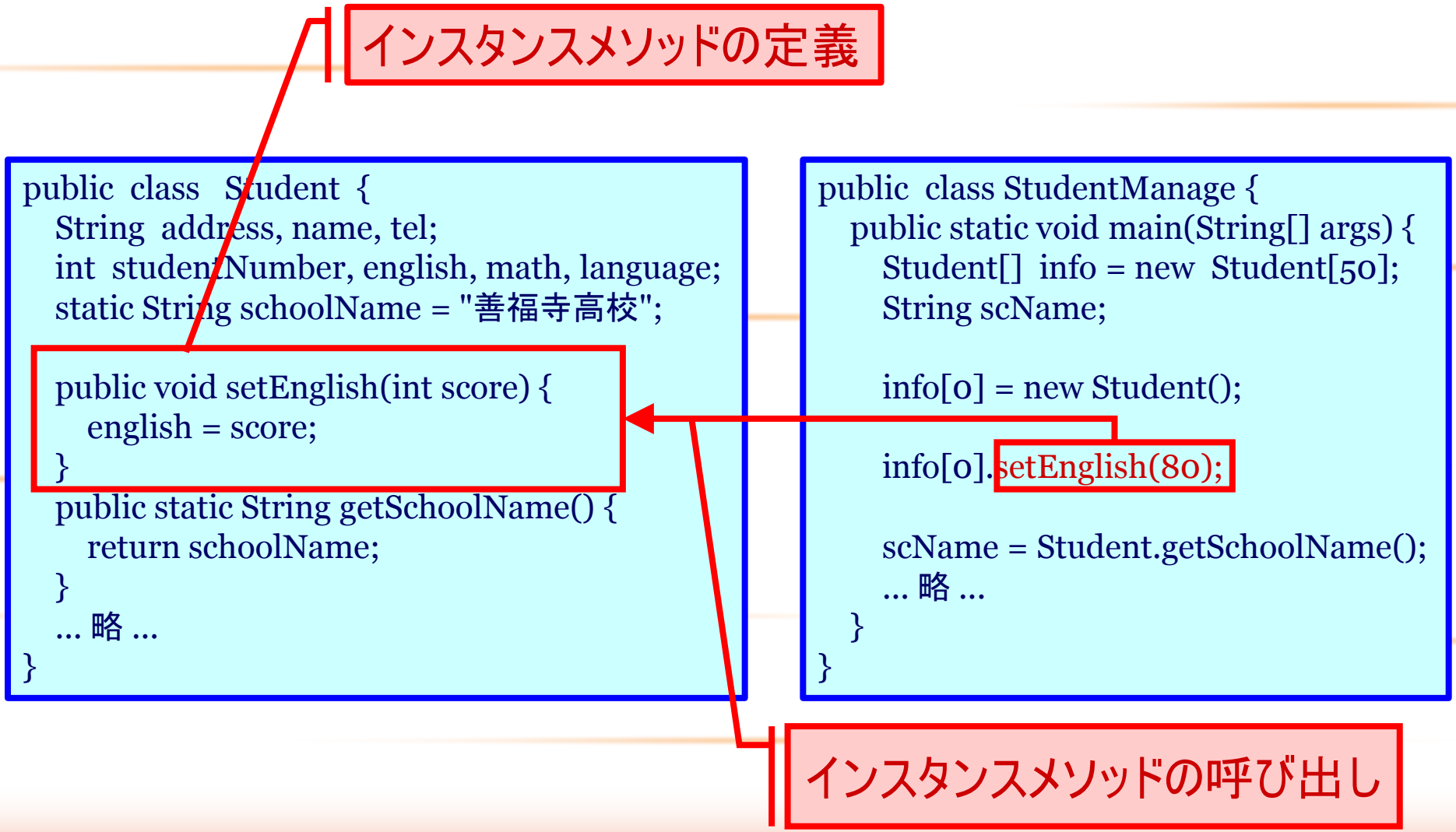
メソッドの呼び出し(例)(1)

```
public  class   Student  {
   String  address, name, tel;
   int  studentNumber, english, math, language;
   static String schoolName = "善福寺高校";

   public void setEnglish(int score) {
      english = score;
   }
   public static String getSchoolName() {
      return schoolName;
   }
   … 略 …
}
```

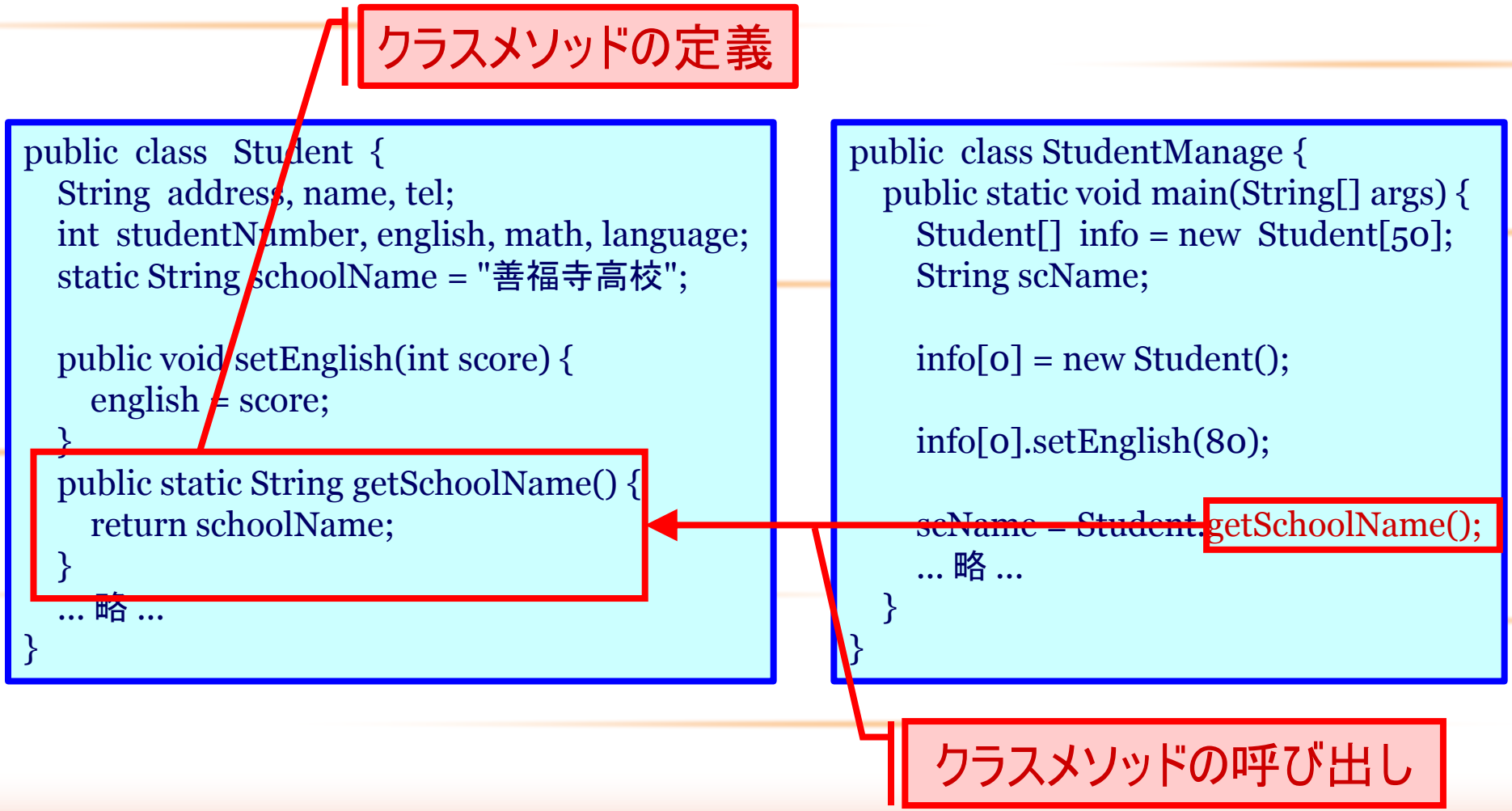
```
public  class StudentManage {
   public static void main(String[] args) {
      Student[]  info = new  Student[50];
      String scName;

      info[0] = new Student();

      info[0].setEnglish(80);

      scName = Student.getSchoolName();
      … 略 …
   }
}
```

※クラス変数は、宣言と同時に値を代入してOK

メソッドの呼び出し(例)(2)



メソッドの呼び出し(例)(3)



継承

継承～「クラス」の親子関係～(1)

■「本」クラスの他に、「雑誌」クラスを考えると…

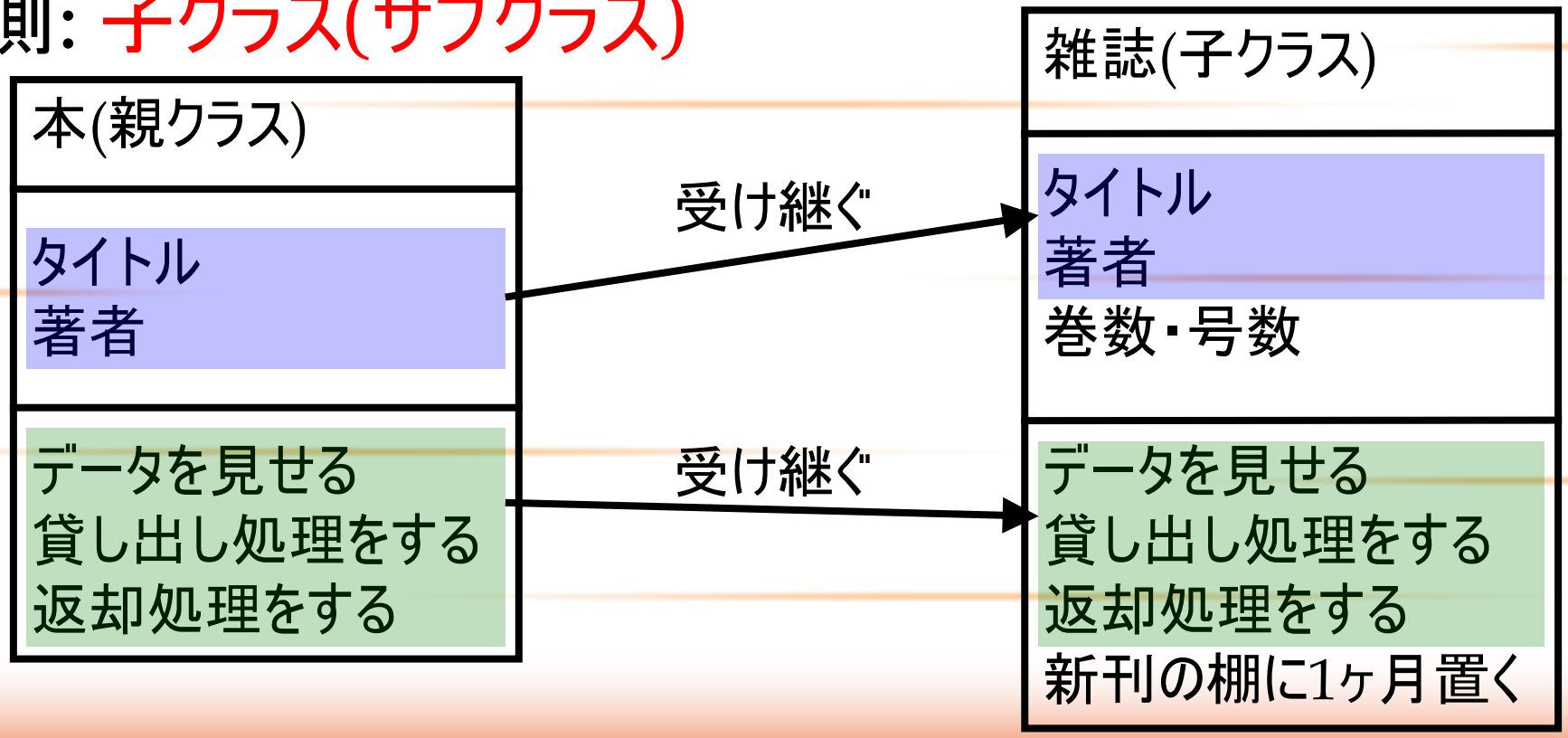
- 「雑誌」クラスは「本」クラスの一部
  - 「雑誌」は「本」でもある(「雑誌」は、「本」の少し細かい分類での名前)
  - 「雑誌」クラスは、「タイトル」や「著者」などの「本」クラスと同じ属性(フィールド)を持つ
  - 「雑誌」クラスは、「データを見せる」、「貸し出し処理をする」などの「本」クラスと同じ操作(メソッド)を持つ

「雑誌」クラスは「本」クラスの性質(属性と操作)をそのまま持っている(受け継いでいる)

あるクラスが別のクラスの性質を受け継いでいるという関係が、「クラスの親子関係」

継承～「クラス」の親子関係～(2)

- 継承: あるクラスが、別のクラスの属性と操作をそのまま受け継いでいるという関係
- 受け継がせる側: 親クラス(スーパークラス)
- 受け継ぐ側: 子クラス(サブクラス)



継承の使い方～拡張～

■クラスAの様々なバリエーションを作りたい場合

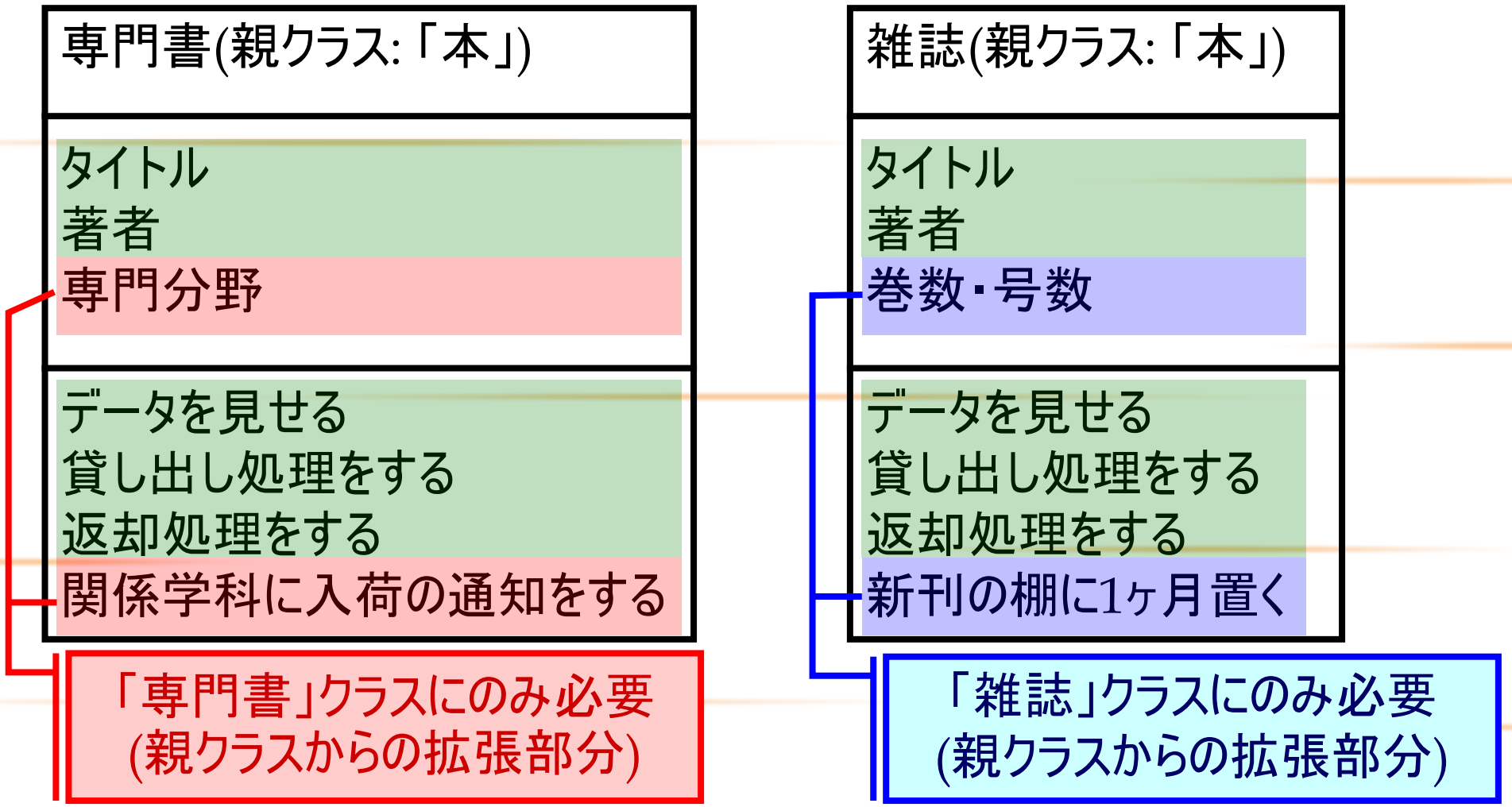
- クラスAを親クラスとして、子クラスB, Cを作る
- クラスBに、Bのための独自の属性や操作を持たせる
- クラスCに、Cのための独自の属性や操作を持たせる

クラスA, B, Cに共通な属性・操作と、クラスB, C独自の
属性・操作を明確に分離できるので、保守性が良くなる

クラスB, C独自の属性や操作をクラスAにまとめて持たせると、クラスAが大きくなって保守性が悪くなる

機能などの変更のときに、どの部分を修正すれば
良いかがわかりやすくなる

継承の使い方～拡張～(例)



親クラス(「本」クラス)から継承した属性と操作(「専門書」・「雑誌」にも必要な共通の属性と操作,「専門書」・「雑誌」クラスでは定義不要)

継承の使い方～抽象化～

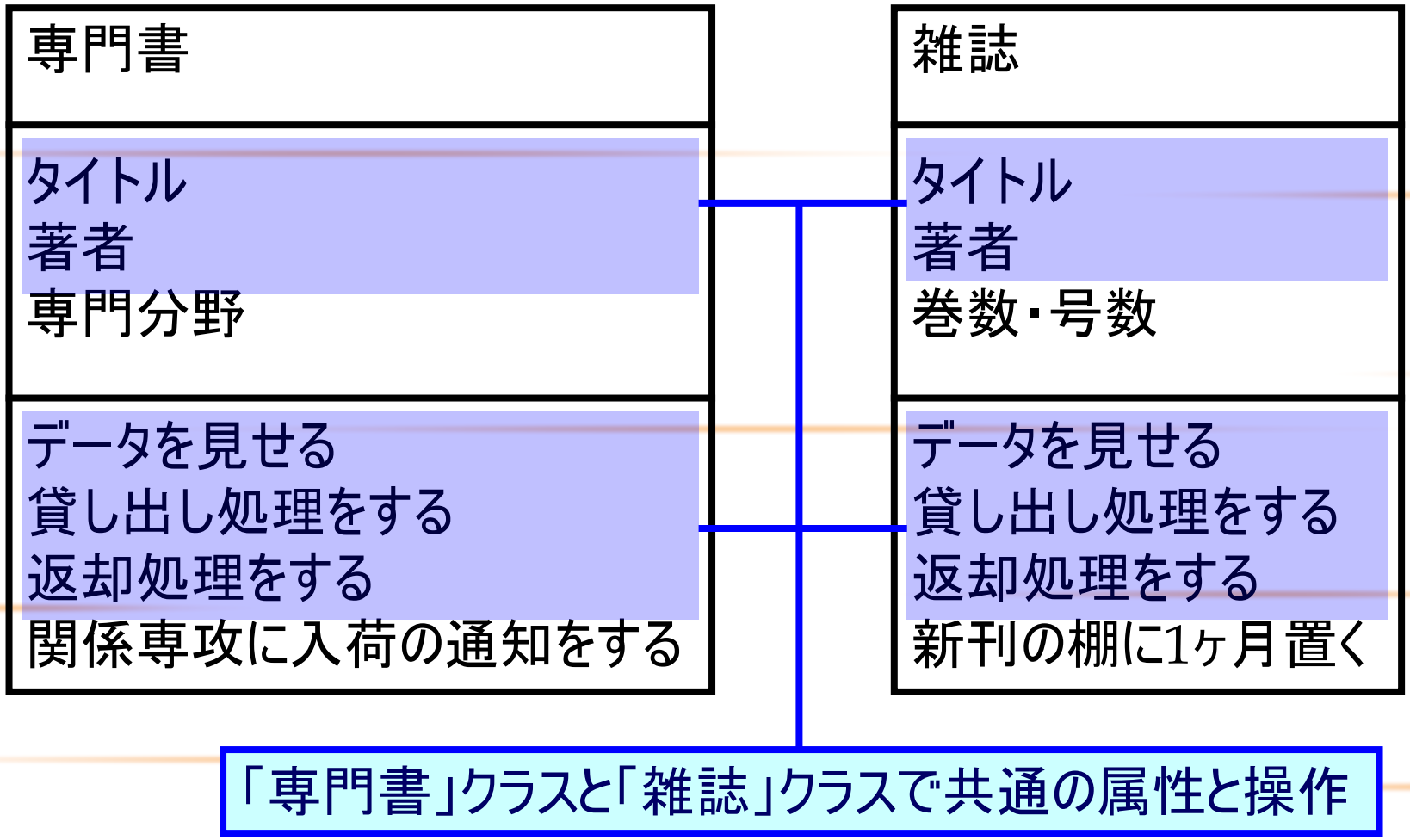
- 共通の属性や操作を持つ複数のクラスの共通部分のみをまとめたい場合
  - クラスB, Cの共通の属性と操作をまとめてクラスAを作る
    - クラスAがクラスB, Cの親クラス
    - クラスB, Cがクラスの子クラス

共通の部分が1箇所にまとまるので、保守性が良くなる

共通の属性や操作をそれぞれのクラスで持っていると、共通部分を変更するとそれぞれのクラスを同じように修正する必要が出てくる

共通する属性や操作を「親クラス」としてまとめると、
共通部分の変更は親クラスのみの修正ですむ

継承の使い方～抽象化～(例)



まとめて「本」クラスを作成すると、変更があっても対処しやすい

継承の使い方～その他～

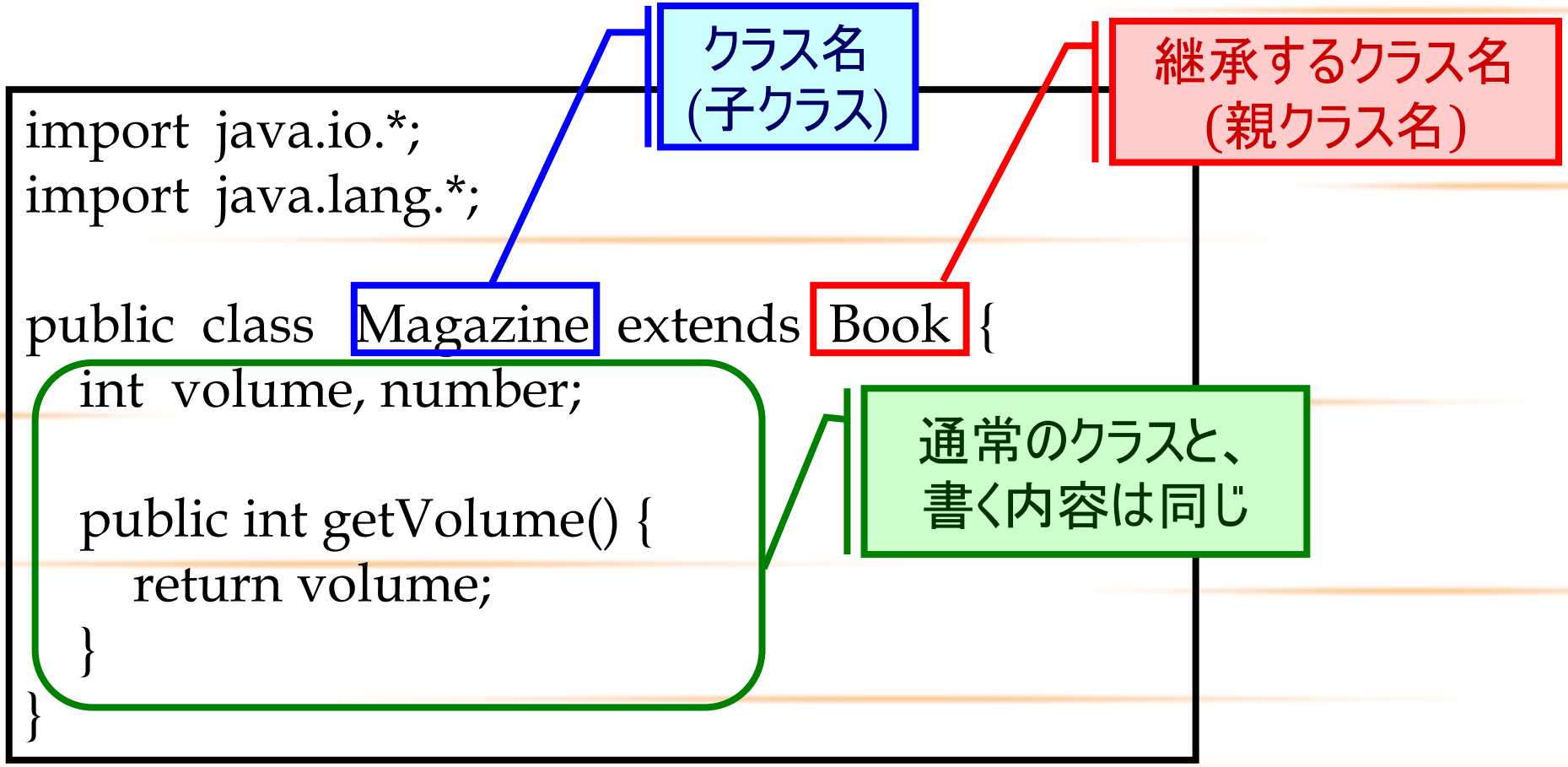
- 子クラスのオブジェクトは、親クラスのオブジェクトとして扱うことも可能
  - 「専門書」クラスのオブジェクトは、「本」クラスのオブジェクトとしても扱うことが可能
    - 「専門書」・「雑誌」などの区別に関係なく共通の処理をしたい場合には、「本」クラスのオブジェクトとして扱う
    - 「専門書」・「雑誌」などで別々の処理をしたい場合には、それぞれのクラスのオブジェクトとして扱う

共通の処理をしたい場合には、子クラスごとに処理を記述する必要がない

プログラムでの継承の表現

継承のしかた

■クラス宣言を記述するときに、クラス名の後に「extends 親クラス名」と書く



継承すると...(1)

■親クラスに定義されているフィールドやメソッドが、子クラスでも自動的に定義されている

継承

```
public  class  Book  {

  String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class  Magazine  extends  Book  {
   int  volume, number;

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class  Library  {
   public static void main(String[] args) {
      Magazine mag = new Magazine();

      mag.title = "今週の東女生";
      mag.author = "東京子";

      String magTitle = mag.getTitle();
   }
}
```

Bookクラスに定義されているものを継承
- 定義しているのはBookクラスにおいてのみ
- Magazineクラスでは実際には記述なし

実際のMagagineクラスの記述

```
public  class  Magazine  extends  Book  {
   int  volume, number;
}
```

継承すると...(2)

■親クラスに定義されているフィールドやメソッドが、子クラスでも自動的に定義されている

継承

```
public  class  Book {

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

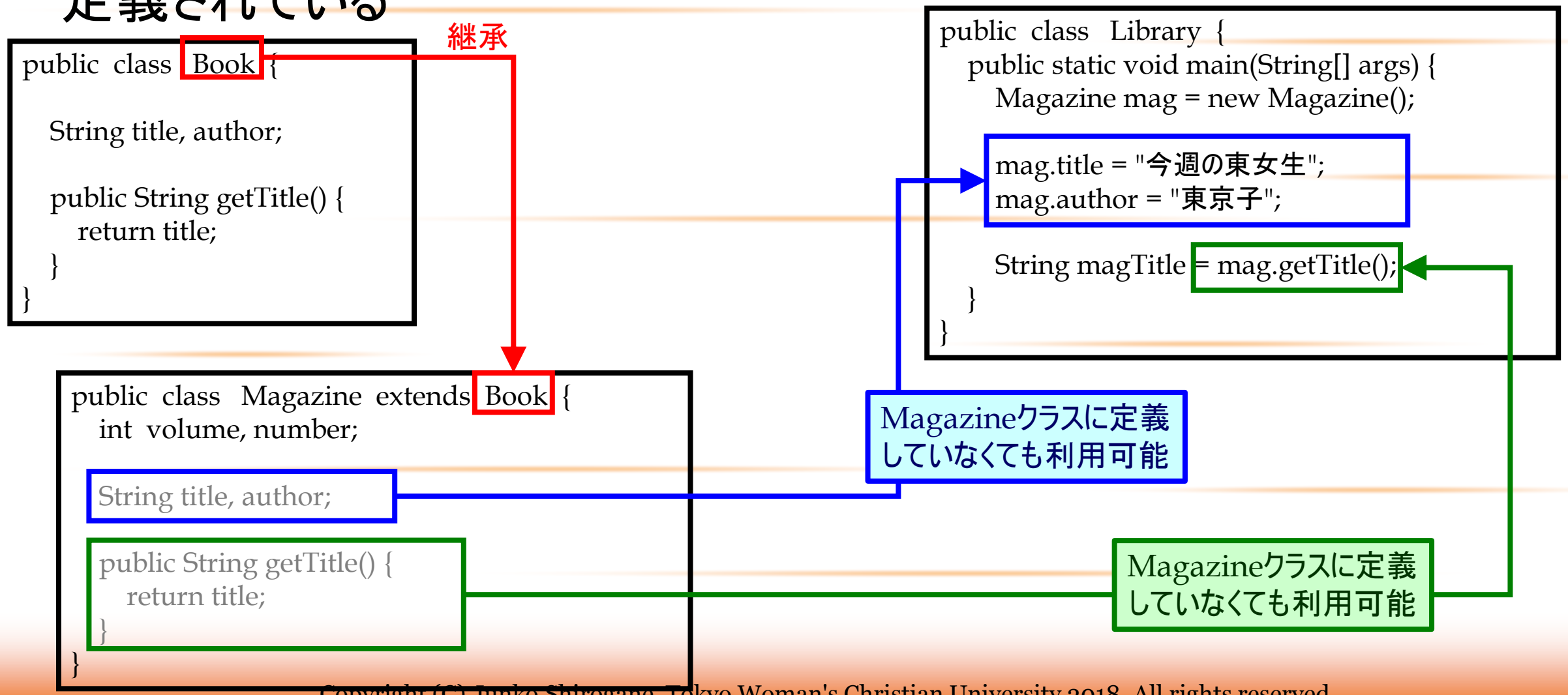
```
public  class  Magazine  extends  Book {
   int  volume, number;

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class  Library  {
   public static void main(String[] args) {
      Magazine mag = new Magazine();

      mag.title = "今週の東女生";
      mag.author = "東京子";

      String magTitle = mag.getTitle();
   }
}
```

Bookクラスに定義されているものを継承
- 定義しているのはBookクラスにおいてのみ
- Magazineクラスでは実際には記述なし

継承すると...(3)

■親クラスに定義されているフィールドやメソッドが、子クラスでも自動的に定義されている



オーバーライド

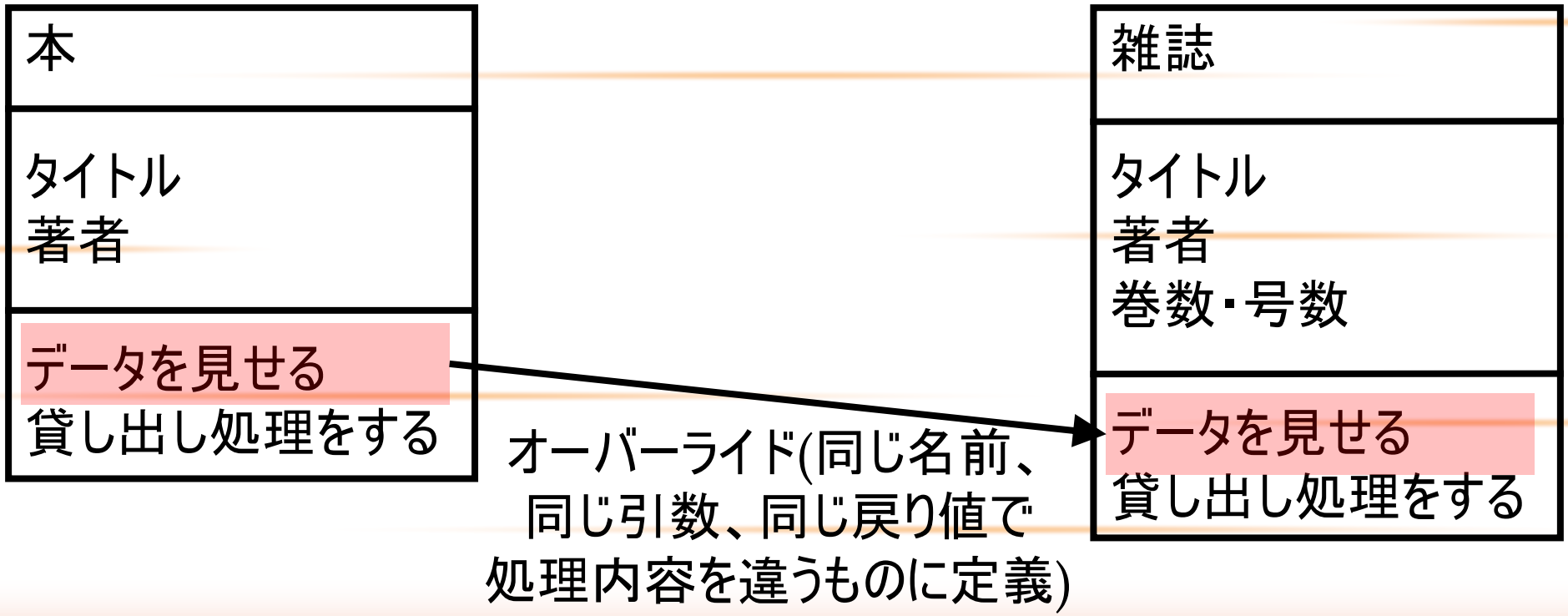
オーバーライド

- 親クラスが持っている操作(メソッド)の処理の内容を子クラスで定義しなおすこと
  - 親クラスが持っている操作(メソッド)は、原則子クラスにそのまま継承(子クラスで全く同じ処理内容で継承)
  - 子クラスで、親クラスとは違う内容にしたい場合に定義しなおすことが可能

親クラスで定義されているメソッドを、
同じ名前と同じ引数、同じ戻り値で、子クラスで定義しなおすこと

オーバーライド(例)

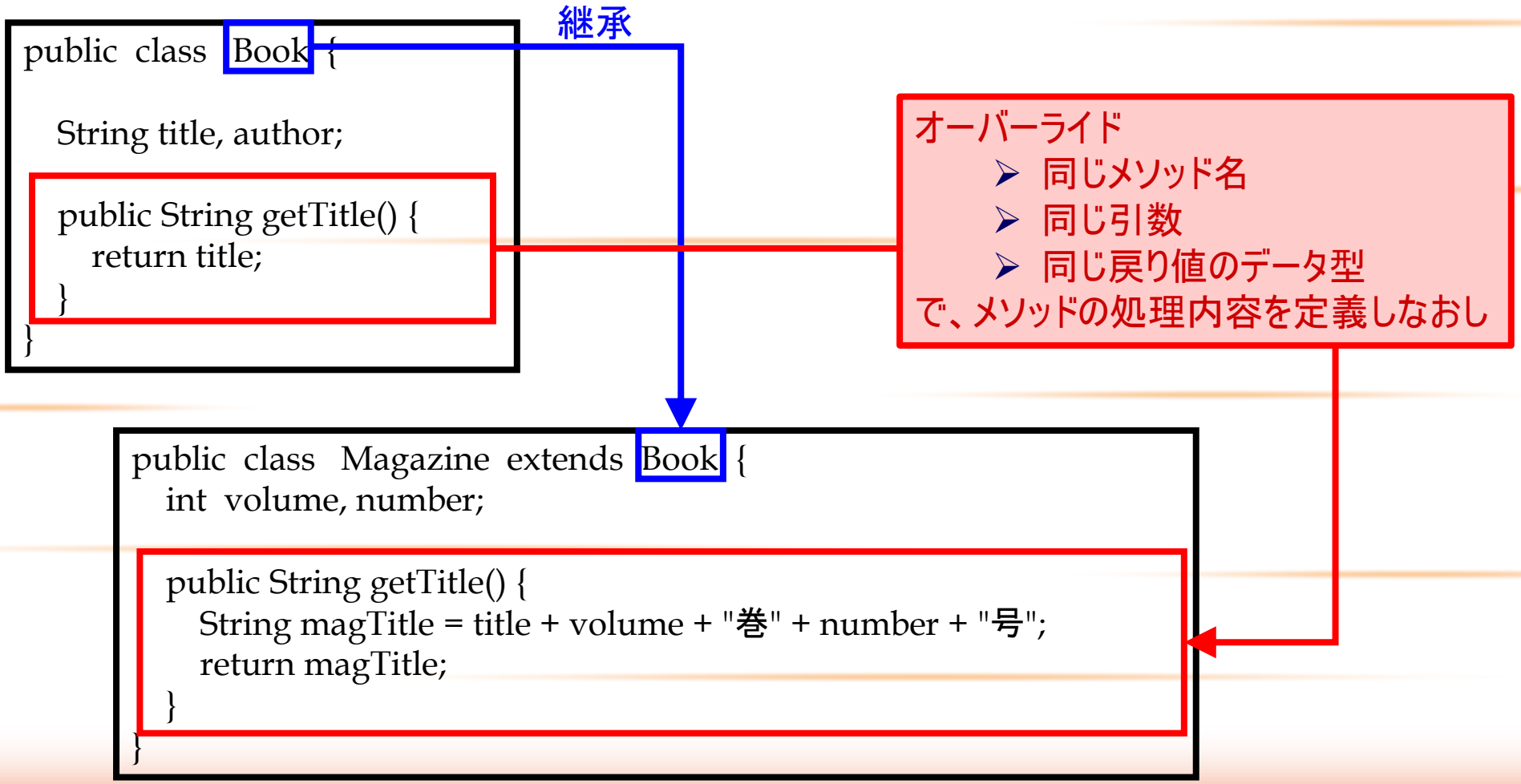
- 「雑誌」クラスでは、雑誌名と巻数を組み合わせて「データを見せる」メソッドの戻り値とする
  - 「データを見せる」メソッドの処理内容を「本」クラスとは違う処理に定義



プログラムでのオーバーライドの表現

オーバーライドするには…

- 親クラスに定義されているフィールドやメソッドが、子クラスでも自動的に定義されている



ポリモーフィズム

ポリモーフィズム(多相性，多態性)

- 同じ名前の操作(メソッド)を異なる形にして呼び出すこと
  - オーバーライドなどによって実現
  - ソフトウェアの実行時になって初めてどの処理が呼び出されるか決定(動的結合)

- 操作の呼び出し側は、どのクラスのオブジェクトの操作を呼び出しているかを意識しなくてすむ
- 操作を呼び出した結果は、それぞれの処理内容に 応じたものになっている

呼び出す操作の内容が違っても、呼び出し側の処理が共通化できる

ポリモーフィズム(例)(1)

- Bookクラスの getTitleメソッドは、本のタイトルがそのまま戻り値
- オーバーライドにより、Magazineクラスの getTitleメソッドは、雑誌のタイトル+巻・号数が戻り値

ポリモーフィズム(例)(2)

■ 子クラスのオブジェクトは、親クラスのオブジェクトとして扱うことも可能

Bookクラスのオブジェクトを作成し、Bookクラスの配列に代入

```
public class Book {
    ... 略 ...
}
```

```
public class Magazine extends Book {
    ... 略 ...
}
```

Magazineクラスのオブジェクトを作成し、Bookクラスの配列に代入(このような処理をしてもOK)

```
public  class   Library  {
   public static void main(String[] args) {
      try {
        String line;
        Book entry[] = new Book[100];
        FileReader fr = new FileReader("BookDatabase.csv");
        BufferedReader br = new BufferedReader(fr);

        while (br.ready()) {
           line = br.readLine();
           if (line.indexOf("Book") == 0) {
             entry[i] = new Book();
           } else {
              Magagine mag = new Magagine();
             entry[i] = mag
           }
        }
      }
      catch(IOException e) {
      }
   }
}
```

ポリモーフィズム(例)(2)

■ 子クラスのオブジェクトは、親クラスのオブジェクトとして扱うことも可能

Magazineクラスのオブジェクトも Book クラスのオブジェクトとして扱うことができる

```
public class Book {
    ... 略 ...
}
```

```
public class Magazine extends Book {
    ... 略 ...
}
```

```
public  class   Library  {
   public static void main(String[] args) {
      try {
        String line;
        Book entry[] = new Book[100];
        FileReader fr = new FileReader("BookDatabase.csv");
        BufferedReader br = new BufferedReader(fr);

        while (br.ready()) {
           line = br.readLine();
           if (line.indexOf("Book") == 0) {
              entry[i] = new Book();
           } else {
              Magagine mag = new Magagine();
              entry[i] = mag
           }
        }
      }
      catch(IOException e) {
      }
   }
}
```

ポリモーフィズム(例)(3)

■ 本のタイトルを調べるとき…

- ただの本を調べる?
- 雑誌を調べる?

} そのときどきによって違う
=プログラムの実行時にしかわからない
=プログラムを作るときには決められない

Ex. ファイルから何冊もの本の情報を読み込んだとき

- Bookクラスの配列に1冊ずつ情報を格納
  →配列の何番目はただの本で、何番目が雑誌かは、プログラム作成時にはわからない

- BookクラスもMagazineクラスもgetTitleメソッドを所有
- MagazineクラスはBookクラスの子クラス

ただの本のタイトルを調べるときも、雑誌のタイトルを調べるときも、
どちらもBookクラスのオブジェクトとして扱えば…?

ポリモーフィズム(例)(4)

- プログラムを作るとき
  - Bookクラスのオブジェクトのget Titleメソッドを呼び出すように、プログラムを記述
- プログラムを実行するとき
  - プログラムは、オブジェクトがMagazineの場合、MagazineのgetTitleメソッドを呼び出し
    - 雑誌のタイトル+巻数・号数という戻り値が返される
  - プログラムは、オブジェクトがBookの場合、BookのgetTitleメソッドを呼び出し
    - タイトルがそのまま戻り値として返される

プログラムを作るときの処理の記述を簡単にできる
(本来は、それぞれで場合分けをして記述する必要がある)

プログラムでのポリモーフィズムの表現

ポリモーフィズムするには...(1)

継承

```
public  class  Book  {

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class  Magazine  extends  Book  {
   int  volume, number;

   public String getTitle() {
      String magTitle = title + volume + "巻" + number + "号";
      return magTitle;
   }
}
```

```
public  class   Library  {
   public static void main(String[] args) {
      Book enrty[] = new Book[100];
      .... 略 ....
      int i;
      for (i = 0; i < 100; i++) {
         System.out.println( entry[i].getTitle() );
      }
   }
}
```

ポリモーフィズムするには…(2)

```
public  class   Book  {

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

- ファイルから本の情報を読み込むなどしている
- 配列entryには、Bookクラスのオブジェクトと Magazineクラスのオブジェクトが入り混じっている

```
public  class   Library  {
   public static void main(String[] args) {
      Book enrty[] = new Book[100];
      .... 略 ....
      int i;
      for (i = 0; i < 1
         System.out.
      }
   }
}
```

例

```
entry[0] = new Book();
entry[0].title = "Javaプログラミング";
entry[0].author = "東京子";

Magazine mag = new Magazine();
mag.title = "月間オブジェクト指向";
mag.volume = 10;
mag.number = 5;
entry[1] = mag;
…
```

```
public  class   Magazine  extends  Book  {
   int  volume, number;

   public String getTitle() {
      String magTitle = title + volume + "巻" + number + "号";
      return magTitle;
   }
}
```

※子クラスのオブジェクトは、一旦子クラスの変数を用意して値を設定し、その後親クラスの変数に代入
(でないと子クラスならではのフィールドに値を設定できない)

ポリモーフィズムするには…(3)

```
public  class   Book  {

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class   Library  {
   public static void main(String[] args) {
      Book entry[] = new Book[100];


                                    ntry[i].getTitle() );
   }
}
```

オーバーライド
- 同じメソッド名
- 同じ引数
- 同じ戻り値のデータ型

で、メソッドの処理内容を定義しなおし

```
public  class   Magazine  extends  Book  {
   int  volume, number;

   public String getTitle() {
      String magTitle = title + volume + "巻" + number + "号";
      return magTitle;
   }
}
```

ポリモーフィズムするには...(4)

実行時: 配列entryにBookクラスのオブジェクトが入っているときに呼び出される

```
public  class   Book  {

   String title, author;

   public String getTitle() {
      return title;
   }
}
```

```
public  class   Library  {
   public static void main(String[] args) {
      Book enrty[] = new Book[100];
      .... 略 ....
      int i;
      for (i = 0; i < 100; i++) {
         System.out.println( entry[i].getTitle() );
      }
   }
}
```

```
public  class   Magazine  extends  Book  {
   int  volume, number;

   public String getTitle() {
      String magTitle = title + volume + "巻" + number + "号";
      return magTitle;
   }
}
```

実行時: 配列entryにMagazineクラスのオブジェクトが入っているときに呼び出される

やってみよう!(1)

- スライドp. 61・p. 67のプログラムを写して、標準出力の結果が本当に違うかどうか確認すること
  - 配列の個数は5個程度で、プログラム内でタイトルや巻・号数を代入すること
    - ファイル入力ではなく、プログラム内で値を入力して良い
  - 配列の1つ1つの要素で、Bookクラスのオブジェクトと Magazineクラスのオブジェクトを作り分けること
    - Ex. 配列の0番目はBookクラス、1番目はMagazineクラス、2番目はBookクラス、…のように分けること
    - Ex. MagazineクラスのオブジェクトのBookクラスへの代入方法

```
Magazine mag = new Magazine();
mag.title = "今週の東女生";
mag.volume = 1;
entry[3] = mag;
```

やってみよう!(2)

- 以下の3つのクラスを持つプログラムを作ること
  - 1つ目：授業の資料のページの「NumberContents.java」をダウンロード
  - 2つ目：NumerContentsクラスを継承したクラス
  - 3つ目：下記の処理をするクラス
    - 標準入力で2つの数を入力し、2つ目のクラスのフィールドに代入
    - 2つ目のクラスで継承しているcalculateメソッドを使い、入力した数の合計を標準出力で出力

2つ目のクラスには、calculateメソッドを定義していないはずなのに、calculateメソッドを使って合計を計算できるかを確認すること

やってみよう!(3)

- 以下の3つのクラスを持つプログラムを作ること
  - 1つ目：授業の資料のページの「NumberContents.java」をダウンロード
  - 2つ目：NumberContentsクラスを継承し、calculateメソッドを
    オーバーライドしたクラス
    - オーバーライド内容：戻り値を2つのフィールドの数の掛け算とするように変更
  - 3つ目：下記の処理をするクラス
    - 標準入力で2つの数を入力し、2つ目のクラスのフィールドに代入
    - 2つ目のクラスでオーバーライドしたcalculateメソッドを使うこと

オーバーライドすることにより、NumberContentsのcalculateメソッドと違う結果(掛け算)が出る
(ポリモーフィズムがきちんとできている)ことを確認すること

やってみよう!(4)

- 下記の3つのクラスを持つプログラム
  - 1つ目：授業のページの「StringModification.java」をダウンロード
  - 2つ目：1つ目のクラスを継承したクラス
    - オーバーライド内容
      - 引数: int型の数を2つ(mとn)
      - 戻り値:「m番目の文字はx、n番目の文字はyです」という文字列(xとyはcharAtメソッドを使って調べること)
  - 3つ目：下記の処理をするクラス
    - 標準入力で入力した文字列を、2つ目のクラスの「text」というフィールド変数に代入
    - 2つ目のクラスで継承しているlengthメソッドを使い、入力した文字列の長さを標準出力で出力

2つ目のクラスには、lengthメソッドを定義していないはずなのに、lengthメソッドを使って長さを調べることができるかを確認すること

やってみよう!(5)

- 下記の3つのクラスを持つプログラム
  - 1つ目：授業のページの「StringModification.java」をダウンロード
  - 2つ目：1つ目のクラスを継承し、substringメソッドをオーバーライドしたクラス
    - オーバーライド内容
      - 引数: int型の数を2つ(mとn)
      - 戻り値:「m番目の文字はx、n番目の文字はyです」という文字列(xとyはcharAtメソッドを使って調べること)
  - 3つ目：下記の処理をするクラス
    - 標準入力で入力した文字列を、2つ目のクラスの「text」というフィールド変数に代入
    - 2つ目のクラスでオーバーライドしたsubstringメソッドを使うこと
      - 引数mとnも、標準入力で入力すること

オーバーライドすることにより、これまで使ってきたsubstringメソッドと
違う結果が出る(ポリモーフィズムがきちんとできている)ことを確認すること