Java演習
第3回

2024/4/24

横山大作

講義前・後の質問

- 横山まで
- dyokoyama@meiji.ac.jp

この資料の内容

- 提出課題2解説

- staticなメンバ
- アクセス修飾子
  - public
- コンストラクタ
- String: クラスの実例
  - メソッドの使い方実例
- 例外
- 典型的なプログラムの書き方

提出課題2（再掲）

- Sequence.main()のメソッドの中に、pointsという配列があるのを確認してください
  - 数列$\{p_i\}$が与えられたと思ってください
- 与えられた数列$\{p_i\}$の隣り合う2つの要素について、以下の処理を行ってください
  - 幅が$p_{i+1} - p_i$、高さが幅の2倍になるような四角形のインスタンスを作りなさい
  - そのインスタンスの面積をarea()というメソッドで計算して、改行付きで出力しなさい

ひな形

```
package javalec2;

public class Sequence {
    public static void main(String[] args) {
        int[] points = {3, 5, 10, 11, 15};
        // ここに課題内容を実装する

    }
}
```

- for文で繰り返し構造を作る
- 中身は
  - Rectangleのクラスを作る
  - 指定された幅、高さにする
  - 表示する

解答例

```
package javalec2;

public class Sequence {
    public static void main(String[] args) {
        int[] points = {3, 5, 10, 11, 15};
        for (int i = 0; i < points.length - 1; i++) {
            int l = points[i + 1] - points[i];
            Rectangle r = new Rectangle();
            r.w = l;
            r.h = l * 2;
            System.out.println(r.area());
        }
    }
}
```

間違えそうなところ

- for文でiはどこまで増やすか？
  - (i+1, i)のペアか、(i, i-1)のペアかでループは違う
  - 結果の数字が何個出るはずか、を見ればデバッグできる
  - 必ず「こうなるはず」を確認しよう！
- インデントの乱れは心の乱れ！
  - インデントが正しくないと、わかってないと思われます
  - 本当にバカに見えるから絶対ダメ！
  - EclipseならSource->Format とか

インスタンスの数

```
for (int i = 0; i < points.length - 1; i++) {
    int l = points[i + 1] - points[i];
    Rectangle r = new Rectangle();
    r.w = l;
    r.h = l * 2;
    System.out.println(r.area());
}
```

```
Rectangle r = new Rectangle();
for (int i = 0; i < points.length - 1; i++) {
    int l = points[i + 1] - points[i];
    r.w = l;
    r.h = l * 2;
    System.out.println(r.area());
}
```

- 動作が違っていることを理解しよう

復習

- クラスとは？
- インスタンスとは？
- メソッドとは？　　言えますか？

メソッド復習

```
class XY {
	public double x;
	public double y;

	public double distance() {
		return Math.sqrt(x * x + y * y);
	}
}
```

- メソッドはオブジェクトにくっついた関数
- メソッドの中ではフィールドの変数が使える
  - インスタンスに保存された値が使える

メソッドの引数

- メソッドの引数に渡すときも、クラスや配列は参照型
  - メソッドの外で作った実体をメソッド内で書き換えられる
- 基本データ型は値がコピーされて渡される
  - メソッドの外の実体は書き換えられない
- 変数への代入と同じような動き

引数と参照型

- (基礎からのJava p.214 問題6)
  - 出力はどうなるか、考えてみよう

```
class Country {
    String name;
    public Country() {
        name = "anonymous";
    }
}

class RefRenshu2 {
  public static void main(String[] args) {
    Country c1 = new Country();

    c1.name = "イギリス";
    changeName(c1);

    System.out.println("name=" + c1.name);
  }

  static void changeName(Country c) {
    c.name = "フランス";
  }
}
```

引数と参照型（その２）

- 基礎からのJava p.215 問題7, 出力はどうなるか考えてみよう
  - これは問題6とよく似てるのに…混乱しがち

```
class RefRenshu3 {
  public static void main(String[] args) {
    Country c1 = new Country();

    c1.name = "イギリス";
    changeName2(c1);

    System.out.println("name=" + c1.name);
  }

  static void changeName2(Country c) {
    c = new Country();
    c.name = "フランス";
  }
}
```

static

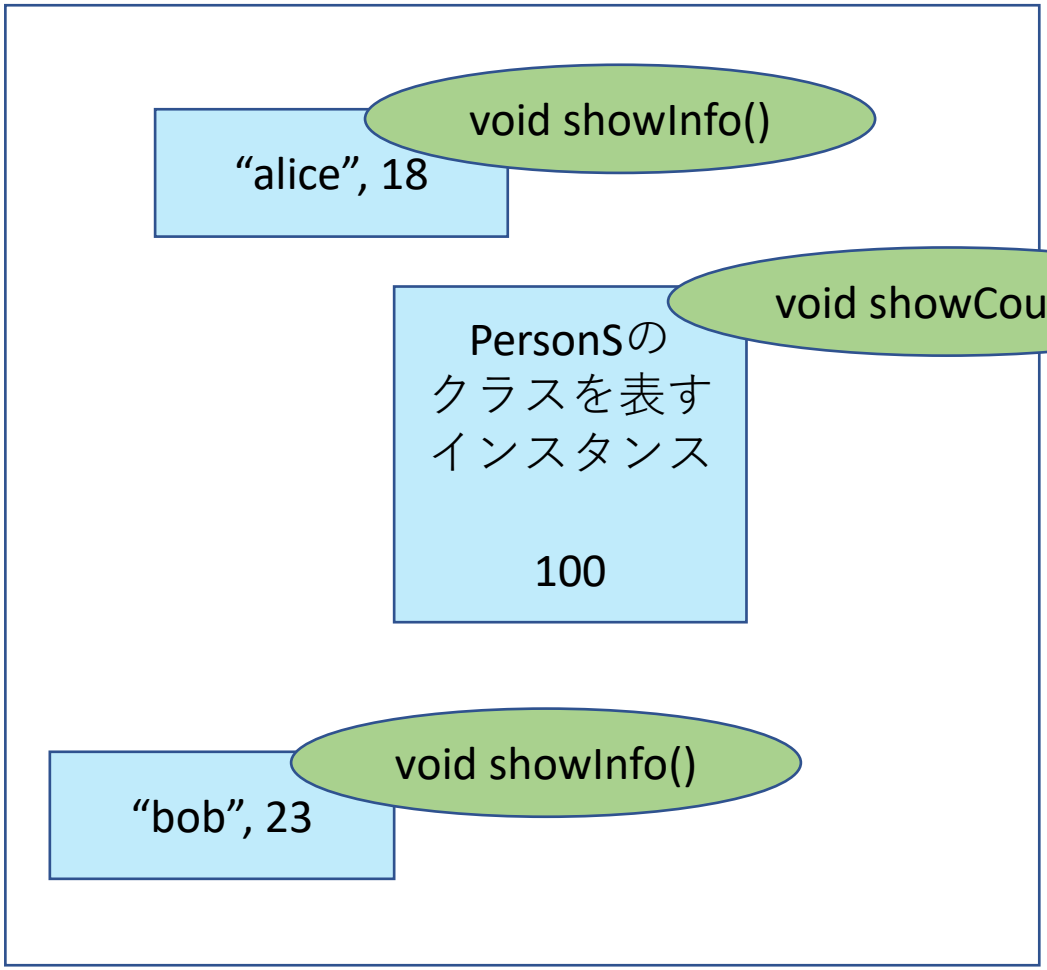
- メソッドは「オブジェクトにくっついて」呼ばれる関数だった
- クラスは「同じデータの仲間をまとめるもの」
  - 学生、座標、…
- 同じデータの仲間を扱う関数をクラスにまとめたいこともある
  - かつ、データの値とは関係ない関数
  - 例: 学生のアカウントに付ける文字”ee”を返す
- このようなメソッドを定義するときは”static”を付ける

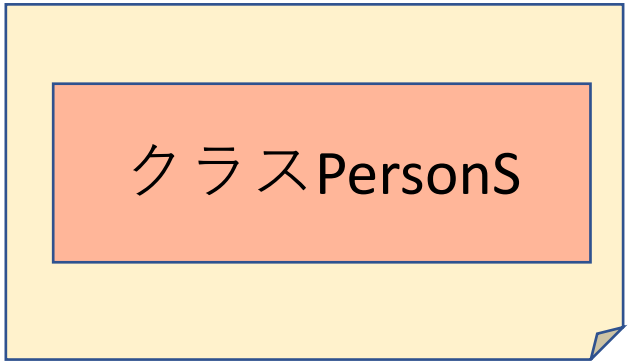
static 修飾子 (see p.510～)

- インスタンスとは別に「クラスに１個、クラスそのものを表すデータがメモリに確保される」というイメージでよい
- メソッドに付ければインスタンスなしで呼べる関数
- フィールドに付ければ「クラスに１個だけある変数」

```
class PersonS {
    static int counter;
    String name;
    int age;

    void showInfo() { ... }
    static void showCount() {
        System.out.println(counter);
    }
}

class StaticSample {
    public static void main(String[] args) {
        PersonS.counter = 100;

        PersonS p1 = new PersonS();
        PersonS p2 = new PersonS();

        PersonS.showCount();
    }
}
```

staticなメンバ





メモリの世界（＝実体）

ソースコードの世界

staticメンバの使い方

- See p.513
- クラス名.メンバ　でアクセスできる

- newしなくても使えていることに注意

```
class StaticSample {
    public static void main(String[] args) {
        PersonS.counter = 100;

        PersonS p1 = new PersonS();
        PersonS p2 = new PersonS();

        PersonS.showCount();
    }
}
```

mainメソッド

- static void main()と定義した
- Javaシステムはオブジェクトは作らずにmainを呼ぶ、と約束したから

public?

- public static void main()のpublicが気になる
- publicは「クラスの外からどう見えるか」を制御する修飾子

クラスの役割の１つ：隠蔽

- 同じ種類のデータをまとめるだけでなく、他の人はその詳細を知らなくてもいいようにしたい

- 例：先生の住所
  - 「情報科学科　xx先生」
  - 実際には部屋に届くのかポストに届くのか、海外出張中なのでスキャンして転送してもらうのか…
  - 外の人にそこまで教えるのは面倒

アクセス修飾子 (see p.472)

- フィールドやメソッドをどこまで見えるようにするか、を制御する仕組み

- public
- なし
- private

くらいがあることを知っておくと良い

  - 詳細はこの先の講義で

- main関数はpublicにしないと呼んでもらえない

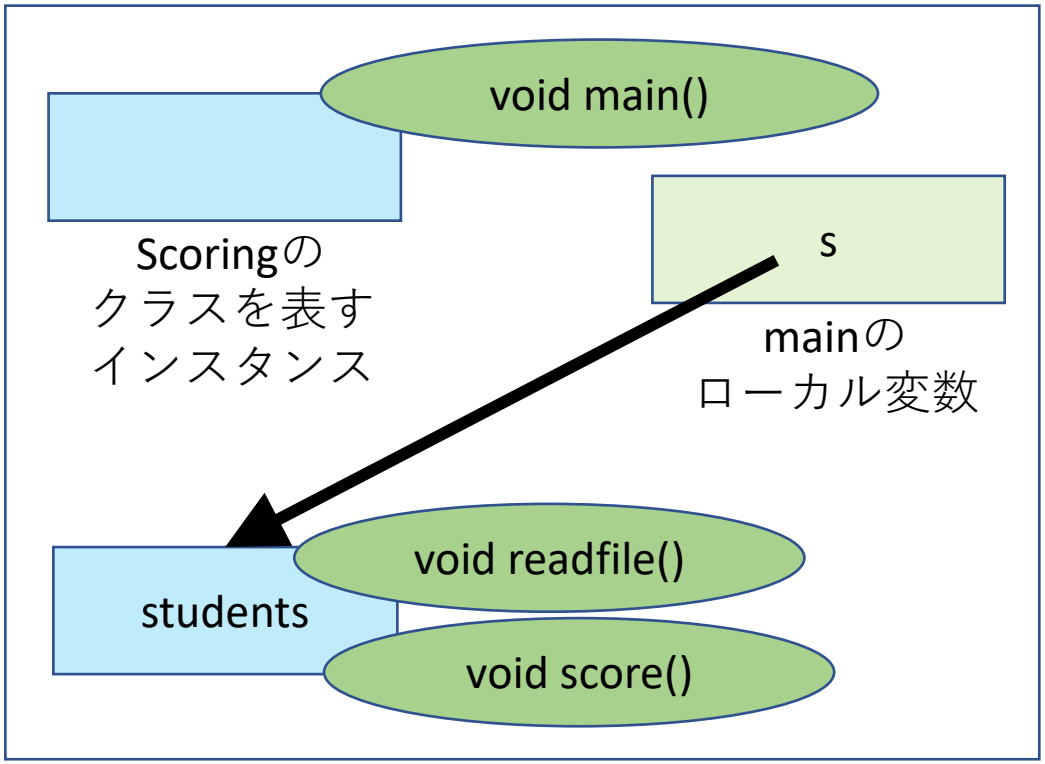
典型的なプログラムの書き方

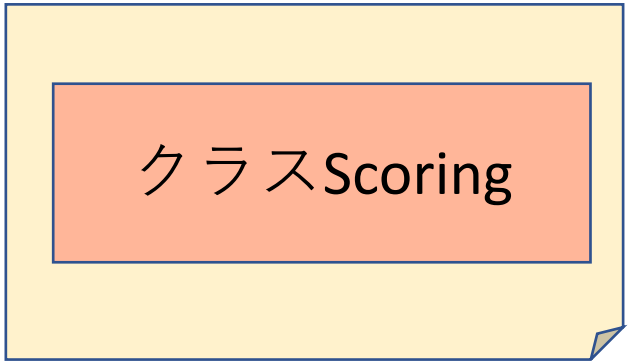
- こんな書き方も多い

```
class Scoring {
  Student[] students;

  void readfile(...) { ... }
  void score(...) { ... }

  public static void main(String[] args) {
    Scoring s = new Scoring();
    s.readfile(...);
    s.score(...);  ... など
  }
}
```

mainの中で自分をnew

- 驚かなくてもじっくり考えれば大丈夫





メモリの世界（＝実体）

ソースコードの世界

アプリケーションとしてのインスタンス

- mainのあるクラスは「プログラムの中心」
- イメージとしては「アプリケーション」「ソルバ」のようなもの
  - Wordを立ち上げる、ブラウザを立ち上げる…
  - 採点をしたい: Scoringアプリを立ち上げる

  - ScoringクラスはScoringアプリを定義しているもの

  - 1つアプリを立ち上げると、アプリが覚えておくべき情報が1かたまり、メモリに作られる
    - メンバ変数: students
    - インスタンスとなる
- mainの中で、自分のインスタンスを1つ作って、そのインスタンスに紐づいた処理をスタートする、という書き方が一般的

コンストラクタ (see p.330)

- インスタンスを作る時、初期化したいことは多い
- 初期化の時にやる処理を「コンストラクタ」として書ける
  - new　したときに呼ばれるメソッドだと思えばよい
- クラス名と同じ名前のメソッド、返り値なし
- 引数を持つこともできる

コンストラクタの例

```
class Person {
  String name;
  int age;

  Person() { //コンストラクタ
    name = “anonymous”; age = 10;
  }

  Person(String iname, int iage) { //これもコンストラクタ
    name = iname; age = iage;
  }

  void showInfo() {
    System.out.println(name + “ “ + age + “歳”;
  }
}
```

コンストラクタの呼び方

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

    Person p1 = new Person();
    p1.showInfo();

    Person p2 = new Person(“Alice”, 10);
    p2.showInfo();

 }
}
```

- newの後ろのクラス名() はこれだった！
- 引数の数や型が違うと違うメソッド（やコンストラクタ）だとJavaは取り扱う
  - 複数の初期化方法が書ける
  - 呼び出されるときには該当するコンストラクタ1つが呼ばれる

使用例

- 課題2でRectangleのwとhを初期化していた
- こういう時こそコンストラクタの出番
  - 四角形は必ずwとhを設定しなくては意味がない、という場合

```
int[] points = {3, 5, 10, 11, 15};
for (int i = 0; i < points.length - 1; i++) {
    int l = points[i + 1] - points[i];
    Rectangle r = new Rectangle();
    r.w = l;
    r.h = l * 2;
    System.out.println(r.area());
}
```

コンストラクタ使えば

```
int[] points = {3, 5, 10, 11, 15};
for (int i = 0; i < points.length - 1; i++) {
    int l = points[i + 1] - points[i];
    Rectangle r = new Rectangle();
    r.w = l;
    r.h = l * 2;
    System.out.println(r.area());
}
```

```
int[] points = {3, 5, 10, 11, 15};
for (int i = 0; i < points.length - 1; i++) {
    int l = points[i + 1] - points[i];
    Rectangle r = new Rectangle(l, l*2);
    System.out.println(r.area());
}
```

コンストラクタの気持ち

- オブジェクトが「壊れた状態」であることをなくす
  - Rectangleのオブジェクトは「必ず意味のある値が入っている」ように強制する気持ちが伝わってくる

```
public class Rectangle {
    int w;
    int h;
    Rectangle(int width, int height) {
        w = width; h = height;
    }
    int area() { return w * h; }
}
```

コンストラクタとオーバーロード

- Javaは関数名が同じでも引数が違うと別のメソッドだと思う
  - 呼ばれた時の引数で、正しい方を選んでくれる
- 「オーバーロード」と呼ぶ
- コンストラクタも同様
  - 引数あり、無しのコンストラクタを両方作って使い分けられる
    - 引数なしなら0で初期化とかが可能

```
public class Rectangle {
    Rectangle() {
        w = 0; h = 0;
    }
    Rectangle(int width, int height) {
        w = width; h = height;
    }
}
```

コンストラクタのオーバーロード

```
public class Rectangle {
    Rectangle() {
        w = 0; h = 0;
    }
    Rectangle(int width, int height) {
        w = width; h = height;
    }
}
```

```
void f() {
    ...
    Rectangle r0 = new Rectangle();
    Rectangle r1 = new Rectangle(3, 5);
}
```

引数なしの方が動く

引数ありの方が動く

デフォルトコンストラクタ (see p.340)

- コンストラクタを定義しないときは、「何もしない」コンストラクタが暗黙のうちに作られる
  - フィールドは決まった初期値（0やnull）が入る

- コンストラクタを1個でも定義すると、デフォルトコンストラクタは消える
  - 自分でコンストラクタを定義すると、今までデフォルトコンストラクタで動いていたところがエラーになったりして困惑しがち

つまり

```
class Person {
  Person(String iname, int iage) { ... }
}

class PersonMain {
  public static void main(String[] args) {
    Person p1 = new Person();
    p1.showInfo();
 }
}
```

- 最初、コンストラクタを作らずに動いていた
  - デフォルトコンストラクタが動いていた
- 初期化がしたくなってコンストラクタ作った
  - 引数付きのものを作った
- コンパイルエラーが起きた
  - デフォルトコンストラクタがなくなるから

クラスの実例

- Stringを例として使い方のイメージをつかんでみよう

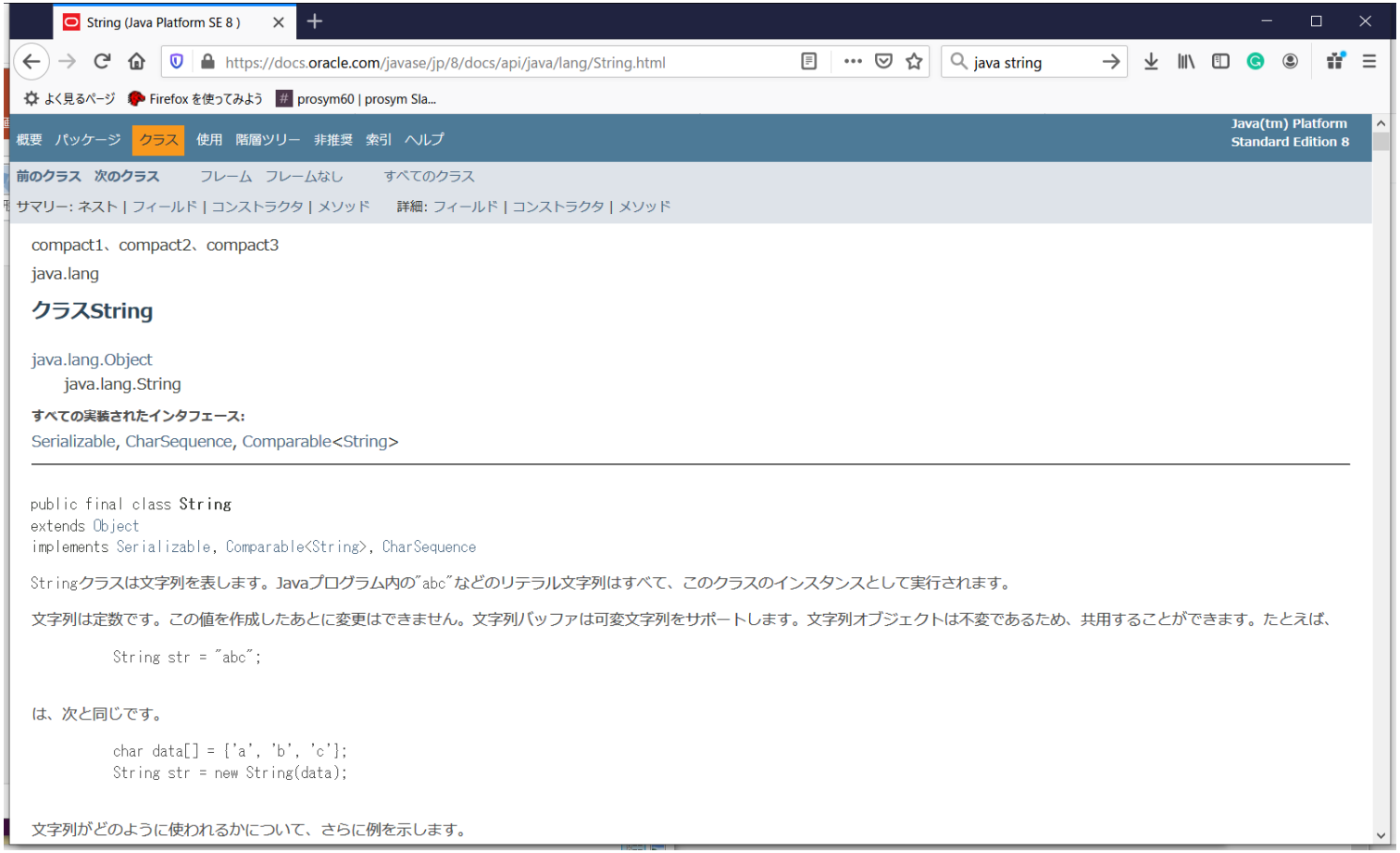
String: 文字列 (See p.327)

- Javaは文字列型がある
- String
- クラス

- 教科書がない人は”java String”くらいでググればoracleのドキュメントが出てくる

- インスタンスを作って使う
- メソッドがある

JavaAPI仕様ドキュメント



- メソッドサマリーあたりを眺めてみよう

Cの文字列との比較

C

- charの配列
- 名前は配列の先頭へのポインタ
- 文字列の終わりが’¥0’というお約束
- 文字列のお約束を知っている関数が多数
  - strlen(char *s);
  - strchr(char *s, int c);

Java

- Stringオブジェクト
- 名前はオブジェクトへの参照
- 中身の詳細、操作はオブジェクトに隠されている
- 基本的な操作のメソッドがくっついている
  - s.length();
  - s.indexOf(int c);

Stringはクラス

- クラスなので「型」になる
- 配列にもできる
- クラスなので、変数に入るのは「参照型」
  - 宣言された時には”null”が入っている
  - 配列は最初は”null”で初期化される
  - 中身が作りたければ別途newしないといけない

```
String s;
s = new String();
System.out.println(s);
```

```
String[] ss;
ss = new String[10];
System.out.println(ss[0]);

ss[0] = new String();
```

ちょっと便利になっている

- 固定された文字列はnewを書かなくても使える
  - こっそりnew String(“文字列”);が行われてインスタンスが作られる
- メソッド以外に便利な演算ができるようになっている
  - +や+=で結合
  - 数字なども結合可能

```
String s = “文字列”;
```

```
s = “文字” + “列”;
s += “abc”;
// s は “文字列abc”に
```

```
int c = 3;
s = “count: ” + c;
// s は “count: 3”に
```

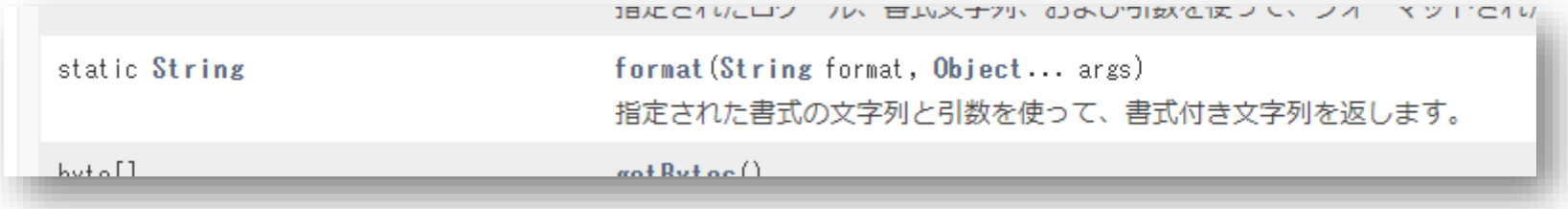
Stringに慣れてみよう

- APIドキュメントには様々なメソッドが書いてある

- 例: replace()

```
String cable = "単位: 本";
String sheet = cable.replace("本", "枚");


System.out.println(sheet);
```

Stringに慣れてみよう



static String	format(String format, Object... args) 指定された書式の文字列と引数を使って、書式付き文字列を返します。
byte[]	getBytes()

- 例: format()

```
double x = 1.0 / 3.0;
String buf = String.format("x: %.3f", x);

System.out.println(buf);
```

  - static と指定されていることに注意
    - staticの場合　String.format　と呼ぶ

文字列の比較 (p. 111)

- 文字列を比較するときにちょっとだけ注意
- if (s == “test”) { … } みたいな比較は「原則ダメ」
- 理由は後日説明します
- 2つの文字列を比較するときは「原則、equals() メソッドを使う」ようにしましょう
- if (s.equals(“test”)) { … }

- ちなみに、switch文も大丈夫(equals()を内部で使っている)

練習

- “abcABCxyzabcxyz”の文字列から、”abc”をすべて見つけて、何文字目から始まるかをすべて表示せよ

- StringのAPIを調べて、使えそうなメソッドは見つけられるだろうか？
  - 提出課題ではないですが、ぜひやってみて。

ヒント

- indexOf()というメソッドは見つかっただろうか？
- 複数のindexOf()があるが、違いはわかる？
  - オーバーロードですね
  - どちらを使う方がいい？

- メソッドを使って、プログラムは書けますか？
  - やってみよう

例外

- Javaに備わるエラー処理の仕組み
- まさに「例外的なことが起こった」ことを示す
- 実行時のエラーで皆様おなじみ

例外の基本的な考え方

- 返り値とは別に、例外という特別な値が返る道を作ろう
  - 関数呼び出し-return　の組とは違う動き
- 関数から返ってくるのは
  - 想定通りの返り値型の値
  - 例外

  のいずれか
- 例外は例外処理用の仕組みで処理しよう
  - 通常処理は返ってきた値を代入して使ったりするが、それとは別の道で処理しよう
  - ここからここまでの例外をまとめて処理、のように楽に書けるような仕組みを提供

例外の発生

argsに要素1個しかないと
“Exception in thread “main”
…ArrayIndexOutOfBoundException..”
みたいなエラーが出る

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

      int result = divOneTwo(args);
      System.out.println(“result=“ + result);



  }
  static int divOneTwo(String[] args) {
    int one = Integer.parseInt(args[0]);
    int two = Integer.parseInt(args[1]);
    return one / two;
  }
}
```

例外はクラス

- オブジェクトが１つ作られて飛んでくる

- どんな例外か
- どこで起きたか（行数とか）
- どんな原因だったか（入力は何だったかとか）

という情報を含んだインスタンスが作られるので、エラー処理やデバッグで使いやすい

例外の基本 (see p.621)

- エラーが起きたところで例外オブジェクトが作られ、投げられる(throw)
- 受け取る(catch)場所に届くまでスタックを上っていく
- main()関数でも受け取られなければその外にデフォルト処理の受け取り方がある
  - エラー表示してプログラムが止まる
  - Exception in thread “main”
    java.lang.ArrayIndexOutOfBoundsException: 0
    at Sample.main(Sample.java: 4)

例外の処理

```
class Sample {
  public static void main(String[] args) {
    try {
      int result = divOneTwo(args);
      System.out.println("result=" + result);
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println(e);
    }
  }
  static int divOneTwo(String[] args) {
    int one = Integer.parseInt(args[0]);
    int two = Integer.parseInt(args[1]);
    return one / two;
  }
}
```

Javaは例外処理を厳しく見る

- 例外処理を忘れてるよ、とコンパイルエラーが出ることも
  - 今の課題程度のプログラムでは出ませんが
- 自分で難しめのプログラム書いていて、コンパイルエラーが例外関係ならこの辺りの書き方を調べよう

- 詳しくはまた後日やるので大丈夫

本日のまとめ

- 提出課題2解説

- staticなメンバ
  - クラスに1つ
- アクセス修飾子
  - public, privateくらいがあることを知っておく
- 典型的なプログラムの書き方
  - アプリケーションのインスタンスをnewする
- コンストラクタ
  - 初期化の書き方
- 例外
  - エラーが出ているときは何かがおかしい