情報処理技法
(Javaプログラミング)1

第5回
コンピュータと情報をやりとりするには? (標準入出力),
エラーメッセージへの対処

人間科学科コミュニケーション専攻

白銀 純子

第5回の内容

- コンピュータとの情報のやりとり
- エラーメッセージへの対処

前回の復習問題の解答

- 下記のプログラムを実行し終わったとき、変数resultの値がいくつになっているかを答えなさい。

```
public class Question4th {
  public static void main(String[] args) {
    int result = 0, apple = 5, banana = 2;

    result = apple * 100 + banana * 200;

    int pine = 1;

    result = result + pine * 250;
  }
}
```

この時点でresultの値は900

結果、900 + 250 = 1150

解答: 1150

コンピュータとの情報のやりとり(入力)

いつもプログラム中に値を書く??(1)(p. 116)

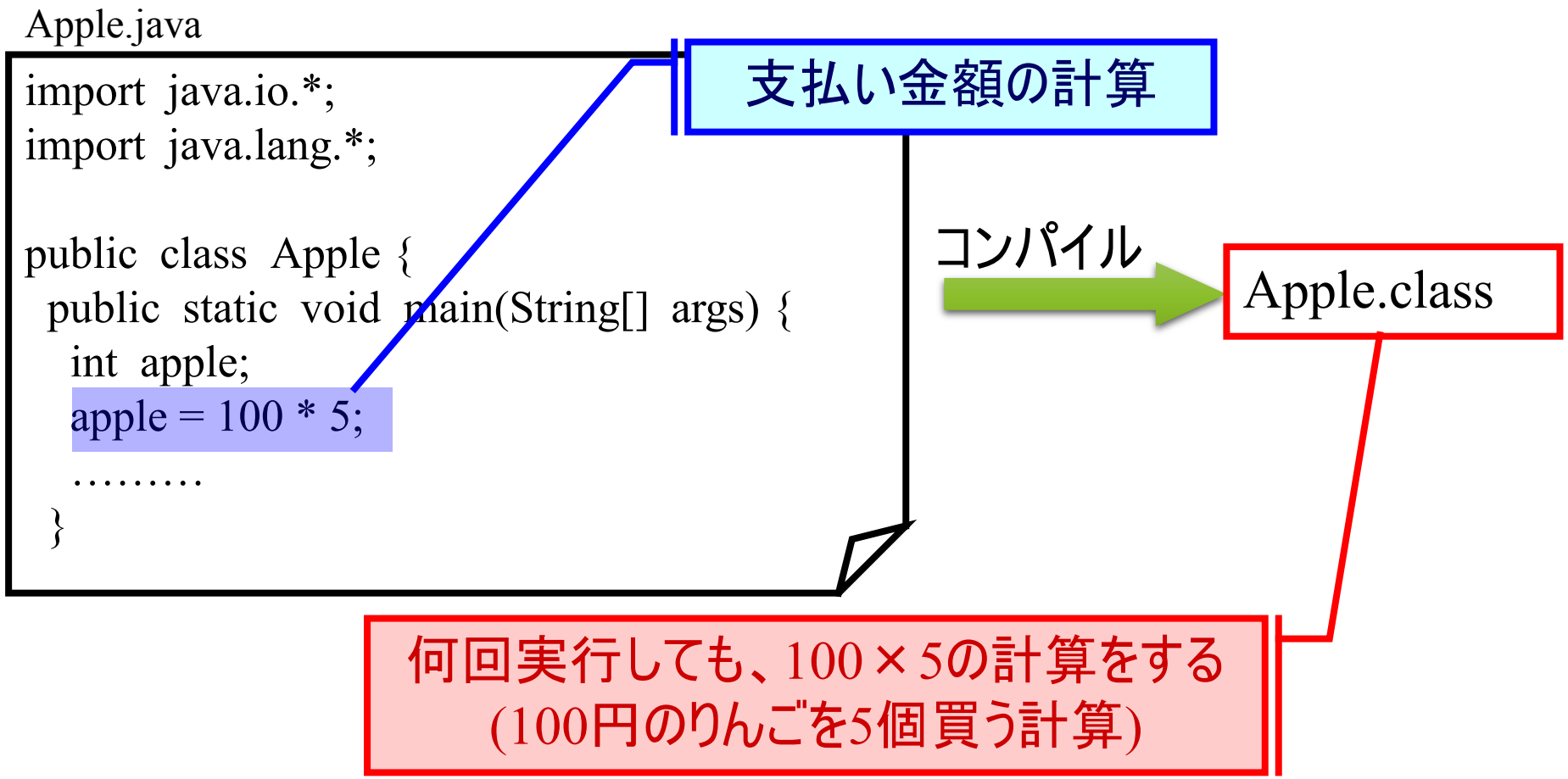
- ここまでの内容: プログラムの中に具体的なデータを書いていた

But...

- スーパーの品物の値段はずっと変わらないものではない
  - 日によって変わることも
  - タイムセールなどがあれば時間によって変わる
- お客が買う品物の種類や個数は人によって違う

いつもプログラム中に値を書く??(2)(p. 116)

- 例えば…100円のりんごを5個買う場合の支払い金額の計算



いつもプログラム中に値を書く??(3)(p. 116)

- 客によっては、1個買う人, 3個買う人, 10個買う人, …
  いろいろな人が存在

Apple.java

```
import  java.io.*;
import  java.lang.*;

public  class  Apple {
  public  static  void  main(String[]  args) {
    int  apple;
    apple = 100 * 5;
    ……
  }
```

レジの人は、客がりんごを持ってくるたびにこの部分を書き換えてコンパイルしなおす???

➡ 面倒で、特に忙しいときはそんなことはやってられない

いつもプログラム中に値を書く??(4)(p. 116)

- 状況によっていろいろ変わるデータは、Javaファイルの中に書きたくない
  - 品物の値段や買う品物の個数など

状況によっていろいろ変わるデータは、
プログラムの外から決めたい

「入力」を利用

値をプログラム外から決めるには?(p. 116)

- GUIで入力する
  - ウィンドウを作って入力フィールドやボタンなどを配置したもの
    (多くのソフトウェアで、入出力に利用されている)
    - ➡ Javaプログラミング2で
- ファイルの中に値を書いておく
  - ➡ できれば、後日この講義で
- 標準入力を使う

「入力」を利用すると...(p. 116)

- プログラムの中(Javaファイル)には、変数を使って計算式だけを書いておく
- その変数に対し、入力されたデータを代入する

Apple.java

```
import  java.io.*;
import  java.lang.*;

public  class  Apple {
  public  static  void  main(String[]  args) {
    int  apple, price, number
    apple = price * number;
    ……
  }
```

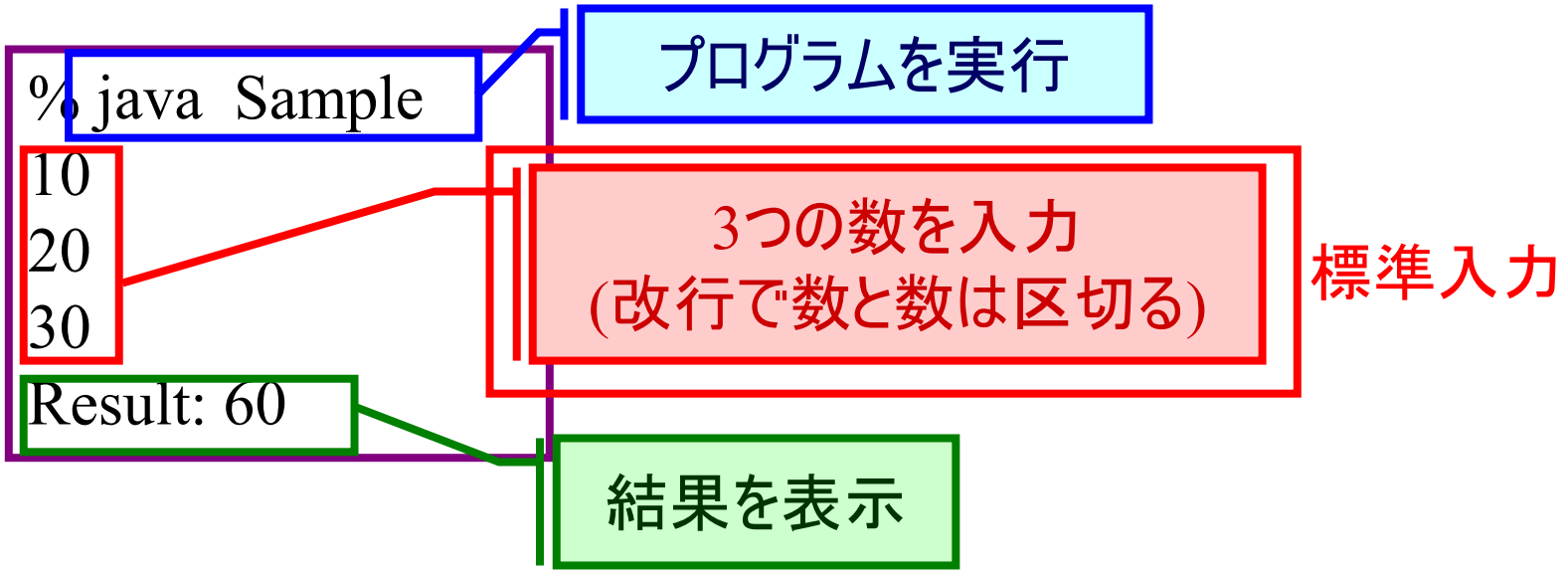
変数を利用した計算式

単価と個数を表す変数には、この計算式の前に、入力されたデータを代入

標準入力とは(p. 119)

- プログラム実行中に、ターミナル上に値を入力し、その値をプログラムが読み取ること

例えば...数を3つ入力して、その合計を求めるプログラム(Sample.java)



ちょっとやってみよう

- 授業の資料のページから、「Average.java」プログラムをダウンロードして実行してみよう!

ターミナルにデータを入力する部分が、標準入力

標準入力のプログラム(p. 119)

```
import java.io.*;
import java.lang.*;
public class Standard {
  public static void main(String[] args) {
    try {
      BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
      String str = br.readLine();
    }
    catch (IOException e) {
      System.out.println("標準入力において例外が発生しました。");
    }
  }
}
```

お約束(1)

ターミナルからの入力の
読み込み部分

お約束(2)

お約束(1)(p. 121)

- try: 例外処理
  - 処理を実行するときに、何か問題が起こるかもしれない場合、「try{ ～}」の間にその処理内容を書く
  - 標準入力の場合、場合によっては入力された文字を読み込めない場合があるため、例外処理をする
  - 「try {～}」の間に、全ての処理内容を書く
- BuffrerdReader: 文字を入力するための準備

try～catchの注意(p. 121)

```
import java.io.*;
import java.lang.*;
public class Standard {
  public static void main(String[] args) {
    try {
      BufferedReader br =
              new BufferedReader(new InputStreamReader(System.in));

      String str = br.readLine();

    }
    catch (IOException e) {
      System.out.println("標準入力において例外が発生しました。");
    }
  }
}
```

tryのカッコを閉じる関係
- catchをtryのカッコの中に書いてはならない

tryの閉じカッコとcatchの間には
何も書いてはならない

標準入力のプログラム(p. 119)

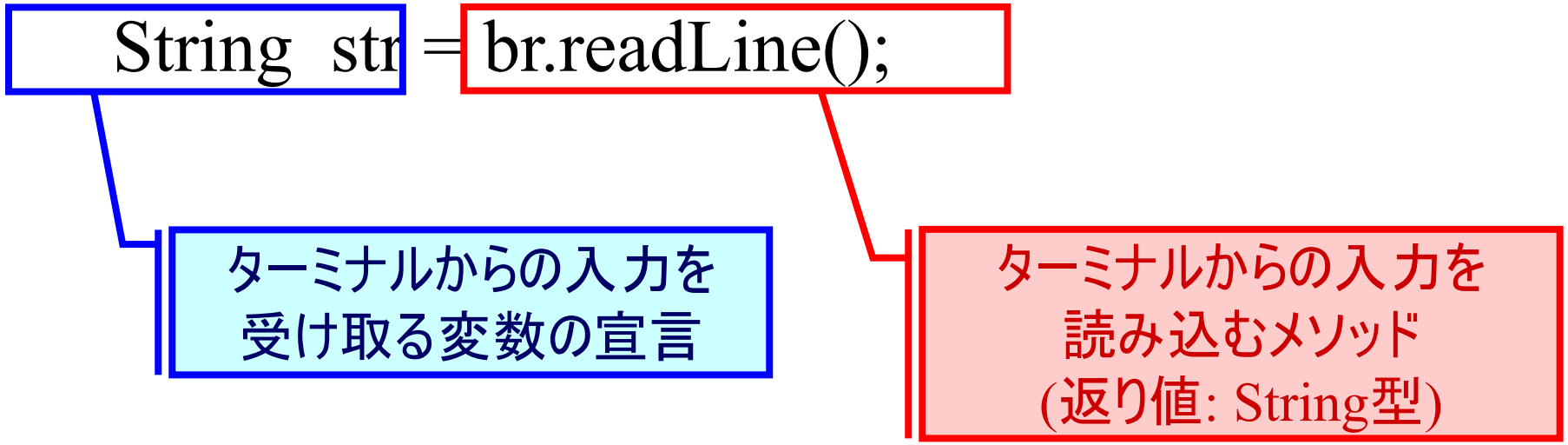
```
import java.io.*;
import java.lang.*;
public class Standard {
  public static void main(String[] args) {
    try {
      BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
      String str = br.readLine();
    }
    catch (IOException e) {
      System.out.println("標準入力において例外が発生しました。");
    }
  }
}
```

お約束(1)

ターミナルからの入力の
読み込み部分

お約束(2)

ターミナルからの読み込み部分(1)(p. 124)

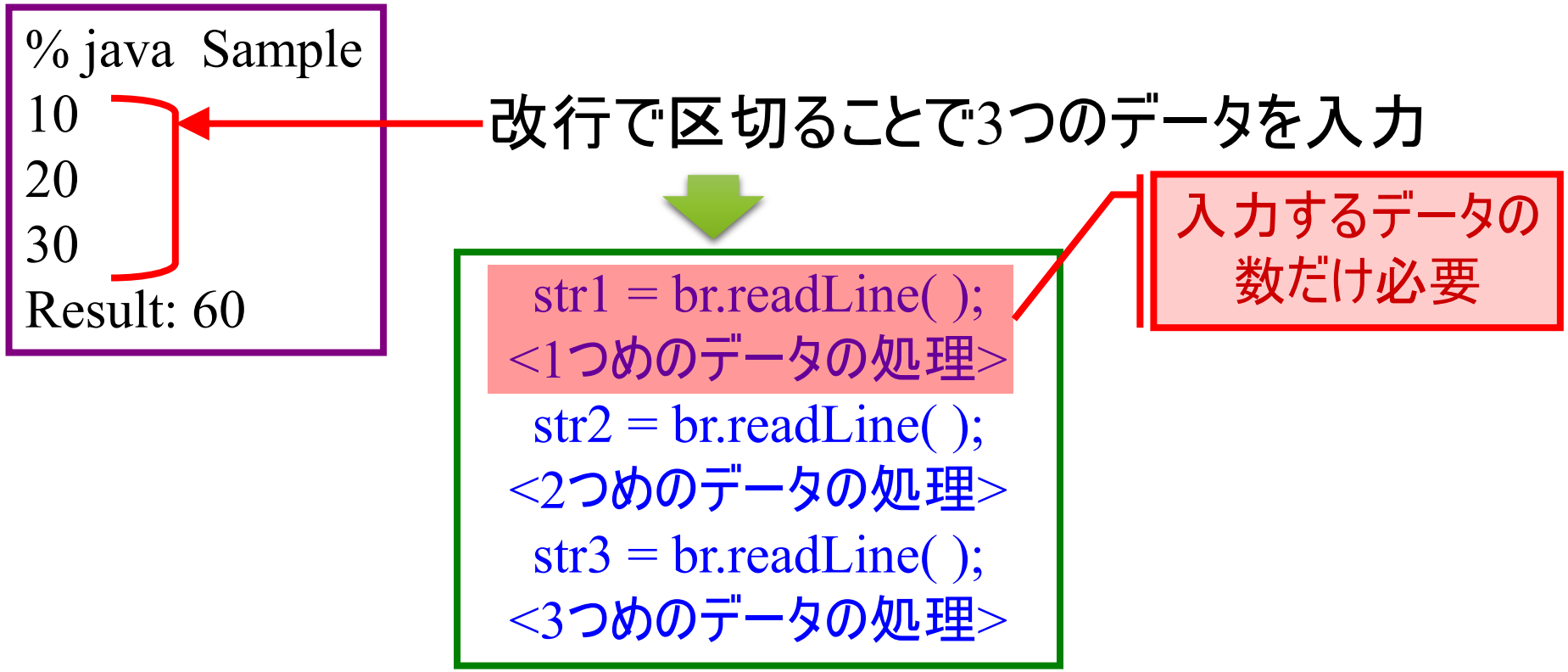


この式では、ターミナルから読み込んだ文字列を「str」に代入している

※「br.readLine()」の「br」は、「BufferedReader br」の「br」
(brでなくても良いが、この2つを合わせる必要)

ターミナルからの読み込み部分(2)(p. 125)

- 「br.readLine( )」で読み込むのは、改行まで(データ1つ分だけ)



ターミナルからの読み込み部分(3)(p. 125)

- 「br.readLine()」で読み込まれるものは、必ず「String」型
  - 数でない文字列を扱うときにはこれでいい
  - 入力されたものが数であっても、コンピュータは「文字の連なり」と考えていて、数値とは考えていない

⇒ 数が入力される場合には、「それは数値である」とコンピュータに教える必要

コンピュータに「それは数値である」と教える方法は?

文字列を数値に変換(int型)(p. 126)

- 文字列を、「それはint型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Integer.parseInt( str );
```

コンピュータが「文字の連なり」と考えているものを、
int型の数値であると教えるメソッド

➡ この結果をint型の変数に代入

つまり... `num = Integer.parseInt(str);`

※「num」はint型の変数

文字列を数値に変換(float型)(p. 126)

- 文字列を、「それはfloat型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Float.parseFloat( str );
```

コンピュータが「文字の連なり」と考えているものを、
float型の数値であると教えるメソッド

➡ この結果をfloat型の変数に代入

つまり... `num = Float.parseFloat(str);`

※「num」はfloat型の変数

文字列を数値に変換(double型)(p. 126)

- 文字列を、「それはdouble型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Double.parseDouble( str );
```

コンピュータが「文字の連なり」と考えているものを、
double型の数値であると教えるメソッド

➡ この結果をdouble型の変数に代入

つまり... `num = Double.parseDouble(str);`

※「num」はdouble型の変数

標準入力のプログラム(p. 119)

```
import java.io.*;
import java.lang.*;
public class Standard {
  public static void main(String[] args) {
    try {
      BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
      String str = br.readLine();
    }
    catch (IOException e) {
      System.out.println("標準入力において例外が発生しました。");
    }
  }
}
```

お約束(1)

ターミナルからの入力の
読み込み部分

お約束(2)

お約束(2)(p. 121)

- catch: 例外処理
  - 「try」を書くと、「try」の閉じカッコの次に必ず書かなければならない
  - 「try」の中の処理で何か問題が起こったときに、「catch(…) {〜}」の処理が行われる
    - Ex. 処理を実行中にエラーが起こった場合に、エラーメッセージを表示する、など
    - 今回のテンプレートでは、標準出力でメッセージを1つ出力
      (標準出力については、後で)

サンプルプログラム

「main(String[] args) {」以下の部分

```
String  str1, str2, str3;
int  num1, num2, num3, result;
try {
  BufferedReader br =
                new BufferedReader(new InputStreamReader(System.in));
  str1 = br.readLine();
  str2 = br.readLine();
  str3 = br.readLine();
  num1 = Integer.parseInt(str1);
  num2 = Integer.parseInt(str2);
  num3 = Integer.parseInt(str3);

  result = num1 + num2 + num3;
}
catch (IOException  ioe) {
  System.out.println("標準入力において例外が発生しました。");
}
```

標準入力から3つのデータを読み込み、str1, str2, str3という変数に格納

標準入力から読み込んだ3つが数値であるとコンピュータに通知

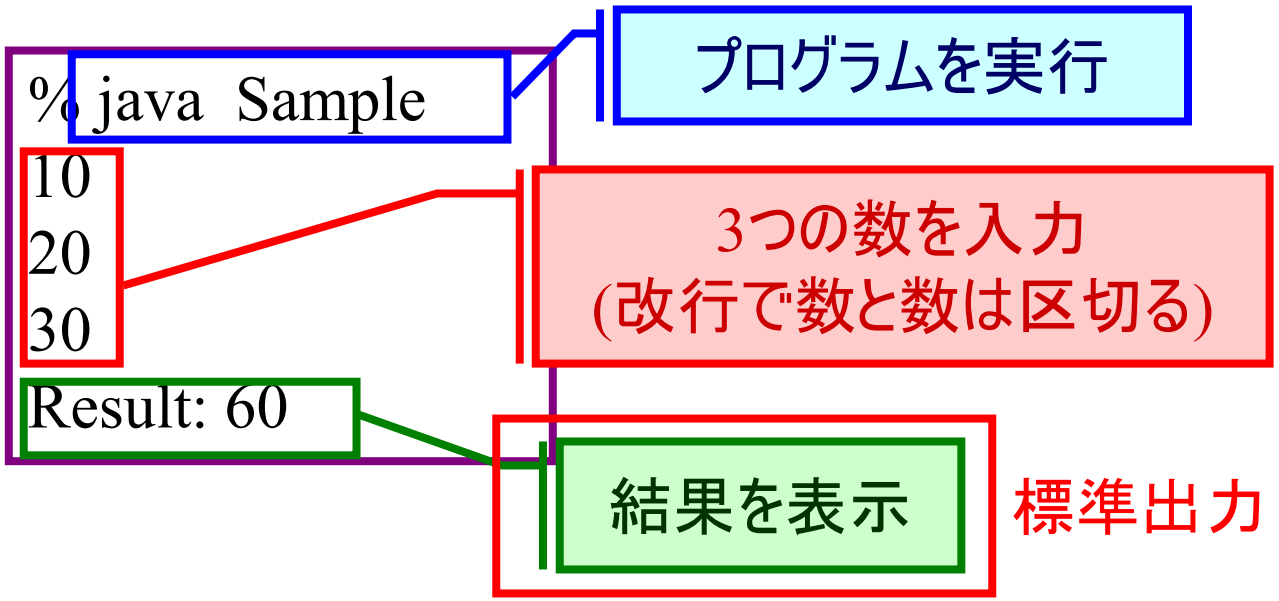
入力されたデータを足し合わせて結果を計算

コンピュータとの情報のやりとり(出力)

標準出力とは(p. 128)

- プログラム実行中に、ターミナル上に何らかの文字列を表示させること

例えば...数を3つ入力して、その合計を求めるプログラム(Sample.java)



標準出力の方法(1)(p. 128)

- System.out.println(*文字列や数値*);
  - ✓ 出力部分(「文字列や数値」の部分)の後に改行が入る
- System.out.print(*文字列や数値*)
  - ✓ 出力部分(「文字列や数値」の部分)の後に改行が入らない

標準出力に処理結果を
表示(出力)する命令

「*文字列や数値*」の部分: String型のデータの作り方と同じ

- 出力したい文字列や変数を「+」でつなげて書く
- 変数でない文字列は、「"」で囲んで書く

標準出力の方法(2)(p. 128)

- 「Please input a number」を表示したい場合
  (全て変数でない文字列)

```
System.out.println("Please input a number.");
```

- 「3 apples」と表示したい場合(「3」は変数「num」の値,
  「apples」は変数でない文字列

```
System.out.println(num + " apples");
```

変数(「"」では囲まない)

変数でない(「"」で囲む)

標準出力の使い方いろいろ(1)(p. 130)

- 標準入力の際に、何を入力すれば良いかの指示を表示
  - 「XXの平均を計算します。数をX個入力してください。」
  - 「品物の金額を計算します。販売する品物の個数を入力してください。」
  - etc.
- 処理の結果(計算結果など)を表示(どのようなものの結果なのかを表す文章などとともに表示)
  - 「平均を計算した結果: ***計算結果***」
  - 「支払い金額: ***計算した金額***」
  - etc.

標準出力の使い方いろいろ(2)(p. 130)

- 正しく処理がされているかどうか(間違いがないか)を確認
  - コンパイルができても、プログラムが正しく動作するとは限らない!
  - プログラム中の計算方法が間違っているかも…

➡ プログラムの処理結果(途中計算の結果)を表示したい!

➡ 標準出力で結果(途中結果が入っている変数の内容など)を表示する

文字列の扱い方の注意(1)

- 「”」で囲まれた単語は、コンピュータは単なる文字列と判断
  - プログラム中の適切な箇所に書かれていれば、コンパイルエラーなし
- 「”」で囲まれていない単語は、コンピュータは変数と判断
  - 宣言しないまま書かれていると、コンパイルエラー
    (「シンボルを処理解釈できません」というメッセージ)

文字列の扱い方の注意(2)

- System.out.println("Please input the number of apples");
  - これから標準入力で何を入力すべきかを指示する文章
  - 単なる文章の表示なので、「"」で囲む必要
- System.out.println("Result: " + result);
  - 計算結果(支払い金額など)の表示
    - 「Result:」は、「ここで計算結果を表示します」程度の、「何を表示しているのか」を表す言葉なので、「"」で囲む必要
    - 「result」は、計算結果が格納されている変数で、この変数に入っているデータを表示したいので、「"」は不要

標準出力の「( )」中に各単語は、変数か変数でないかをよく考えること

ちょっとやってみよう

- 授業の資料のページから、「AverageAnother.java」プログラムをダウンロードして実行してみよう!

プログラムの主目的の処理(平均の計算)は、Average.javaと同じ

- プログラムを使う立場だと、Average.javaとAverageAnother.javaのどちらがいいか??

エラーメッセージへの対処法(p. 49)

プログラミングでのエラー(p. 49)

- プログラム作成時に、エラーでうまくいかないことも多い
  - コンパイル時に表示されるエラー: コンパイルエラー
    - スペルミスをした
    - カッコを開き忘れ・閉じ忘れた
    - 必要な場所に必要な命令を書いていなかった, etc.

プログラム中の文法間違い、という意味のエラー

  - コンパイル後、実行時のエラー: 例外
    - 数を0で割ろうとした
    - 使ってはならない番号を使おうとした(配列など), etc.

プログラムに文法間違いはないが、何らかのミスでそれ以上実行できない、という意味のエラー

エラーメッセージの表示(p. 49)

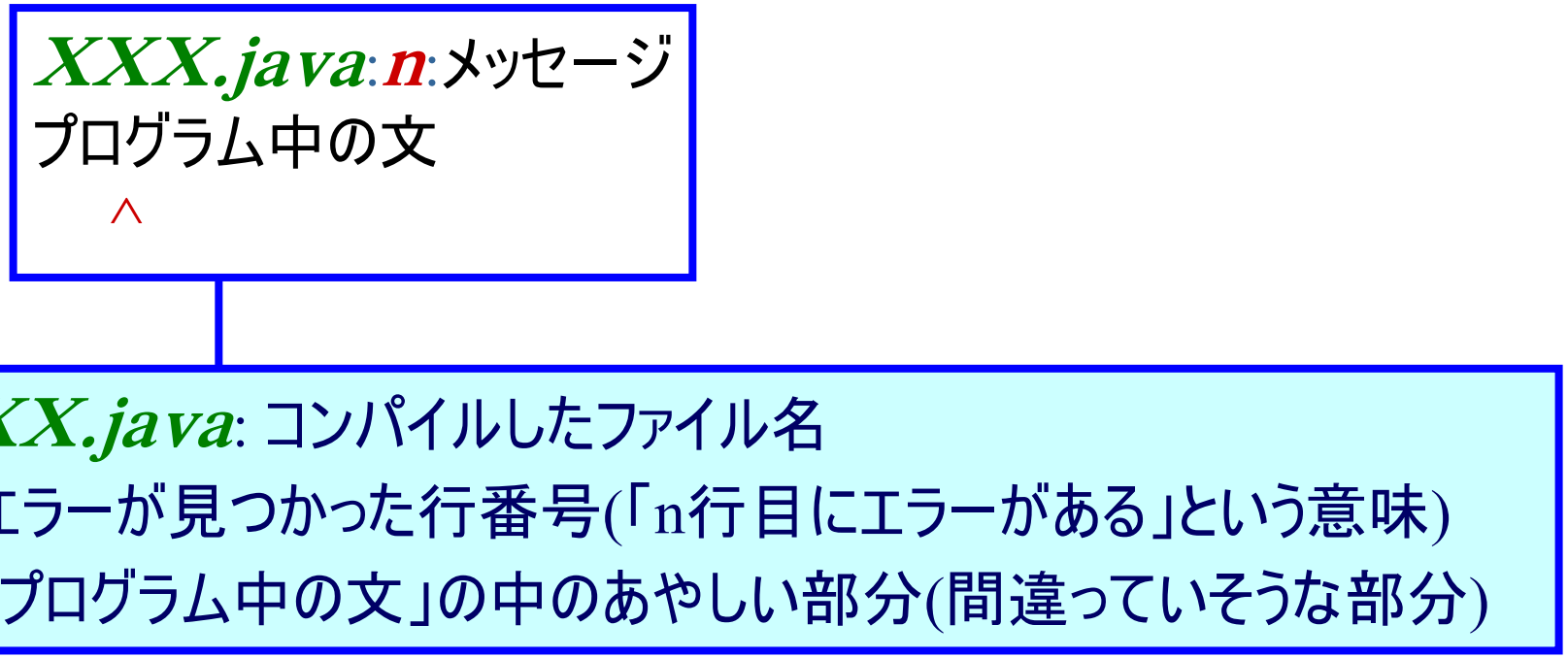
- エラーメッセージは、ターミナル上に表示される
  - コンパイル・実行後に、ターミナルに、予期しないメッセージが表示されていたら、それをよく読むこと
- コンパイルエラーは一度にたくさん表示されることがある
  - 何もメッセージが表示されず、プロンプトが戻ってきたときは、コンパイルが成功
  - メッセージが表示された場合は、コンパイルに失敗

エラーに対処するために... (p. 49)

- Jeditの設定で、行番号と全角スペースを表示するようにしておくと便利
  - 行番号: メニューバーの「表示」→「行番号」→「パラグラフ」にチェック
    - 「パラグラフ」にしておかないと、エラーの行番号(ターミナルに表示される)とJeditでの行番号がずれる
  - 全角スペース: メニューバーの「表示」→「不可視文字の表示」→「全角スペース」にチェック
    - まちがえて全角スペースを書いてしまってエラーになることもよくある

コンパイルエラーの基本形(p. 49)

基本的なコンパイルエラーのメッセージの形



コンパイルエラーへの対処の基本(p. 49)

- コンパイルエラーには一番上から順に対処すること
  - コンパイルエラーがたくさん出てきたときは、多くの場合、上の方に出ているメッセージがより適切な意味
  - 1つのまちがいが影響していろいろな部分のメッセージを出すことも
    - 例えば、宣言していない変数を5箇所で使っていたら、5つエラーメッセージが出てくる
- 「メッセージ」の部分をよく読み、エラーの意味を理解すること
- Jeditで、エラーが出た行番号のところをよく見て、ミスを探すこと

よくあるコンパイルエラー

- コンパイルエラー
  - スペルミス(データ型・変数など)
  - 変数の初期化のし忘れ
  - カッコの対応関係間違い(閉じ忘れ・閉じる場所間違い)
  - ファイル名・クラス名のルールに非準拠
  - try-catchの関係

try-catchの関係(1)(p. 138)

- コンパイルエラー: 'catch' への 'try' がありません

```
public static void main(String[] args) {
try {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("文字列を入力してください。");

String str = br.readLine();
System.out.println("入力: " + str);
catch(IOException e) {
}
}
}
```

カッコの対応関係

catchがtryの中に入ってしまっている

catchをtryの外に出すことが必要

try-catchの関係(2)(p. 135)

- コンパイルエラー: 'try' への 'catch' または 'finally' がありません
- コンパイルエラー: 'catch' への 'try' がありません

```
public static void main(String[] args) {
try {
BufferedReader br =
      new BufferedReader(new InputStreamReader(System.in));

System.out.println("文字列を入力してください。");

String str = br.readLine();
}
System.out.println("入力: " + str);
catch(IOException e) {
}
}
```

tryとcatchの間に文が入ってしまっている

tryとcatchの間の文を適切な場所に移動する必要

try-catchの関係(3-1)(p. 138)

- コンパイルエラー: 例外 java.io.IOException は対応する try 文の本体ではスローされません

```
public static void main(String[] args) {
try {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("文字列を入力してください。");

String str = "abc";
System.out.println("入力: " + str);
}
catch(IOException e) {
}
}
```

try-catchの関係(3-2)(p. 138)

- try～catchでの例外処理は、例外が発生するかもしれない処理をtryの中に書く必要
  - 標準入力で、例外が発生するかもしれない処理: readLine()メソッド
    - readLine()メソッドにより、標準入力が実行
    - 何らかの理由で標準入力ができないとき、readLine()メソッドが実行不能 → 例外発生
  - 例外が発生するかもしれない処理がtryの中に書かれていなければ、コンパイルエラー

try-catchの関係(3-1)(p. 138)

- コンパイルエラー: 例外 java.io.IOException は対応する try 文の本体ではスローされません

```
public static void main(String[] args) {
try {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("文字列を入力してください。");

String str = "abc";
System.out.println("入力: " + str);
}
catch(IOException e) {
}
}
```

標準入力の処理が不要
= readLine()メソッドが書かれていない

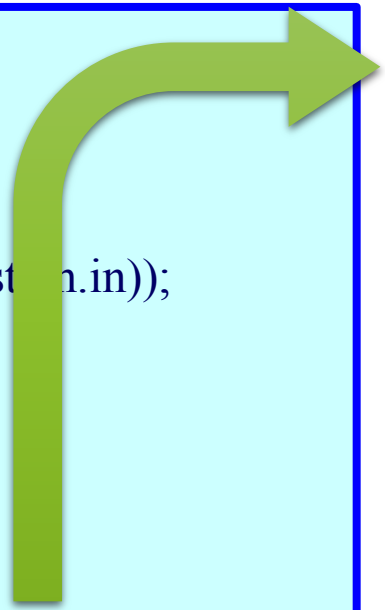
try～catchの文を削除するか、標準入力の処理をする必要

変数の初期化

- コンパイルエラー: 変数が初期化されていない可能性があります

```
public static void main(String[] args) {
String str;
try {
BufferedReader br =
    new BufferedReader(new InputStreamReader(Syst n.in));

System.out.println("文字列を入力してください。");
str = br.readLine();
}
catch(IOException e) {
}
System.out.println("入力: " + str);
}
```



tryの処理の後に実行される

↓

tryの処理は問題が起こったときには実行されない

↓

変数strに値が入らない

↓

この処理の実行時にstrには値が入っていない可能性がある

catchの後の文をtryの中に移動

変数の初期化

- コンパイルエラー: シンボルを処理解釈できません

```
public static void main(String[] args) {
try {
String str;
BufferedReader br =
    new BufferedReader(new InputStreamReader(Syst  .in));

System.out.println("文字列を入力してください。");
str = br.readLine();
}
catch(IOException e) {
}
System.out.println("入力: " + str);
}
```

変数strの宣言がtryの中で行われている

↓

strはtryの中でしか使えない
(詳細は後日)

↓

この処理の実行時にstrの
変数宣言がされていない

catchの後の文をtryの中に移動

例外の基本形(p. 51)

基本的な例外のメッセージの形

Exception in thread "main" ***例外の内容***
　　at 例外の発生場所(***XXX.java***:***n***)

例外の内容:発生した例外の意味(例外の名前)
XXX.java: 実行したファイル名
n: 例外が見つかった行数(「n行目に例外がある」という意味)

例外への対処の基本(p. 51)

- 例外は、1度に1つだけしか表示されない(例外が出るとそこでプログラムの実行が終わってしまうため)
  - 何行もたくさん表示されることがあるが、発生した例外は1つだけ
  - 何行もメッセージが表示されたとしても、「例外の内容」を必ず確認すること
    - 何の例外が起こったのかを、きちんと理解すること
  - 例外が発生した行番号は、多くの場合、「at 例外の発生場所」の1つ目が適切
    - 1つ目の「at 例外の発生場所」を確認し、Jeditでその行番号の処理をよく確認して修正すること

よくある例外

- 例外
  - mainメソッドのスペルミス
  - 文字列のインデックスの取り扱いミス
  - 数値への変換ミス

インデックスの取り扱いミス(p. 109)

- 例外: java.lang.StringIndexOutOfBoundsException

```
public static void main(String[] args) {
String longString = "abcdef", shortString;

shortString = longString.substring(3, 10);
System.out.println(shortString);
}
```

もとの文字列(longString)は、インデックスが5までしかないのに、インデックス10を指定している
=利用可能なインデックスの範囲を超えている

利用可能なインデックスの範囲を確認する必要
(特に標準入力などで文字列を入力した場合)

数値への変換ミス(p. 140)

- 例外: java.lang.NumberFormatException

```
public static void main(String[] args) {
try {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("名前を入力してください。");

String name = br.readLine();
int num = Integer.parseInt(name);
System.out.println("名前: " + name);
}
catch(IOException e) {
}
}
```

「名前」として入力されたデータは通常、数値ではない
= 数値にできない文字列を、数値に変換しようとしている

数値への変換が必要なデータか否かを確認する必要

エラー時のメッセージ

- 個々のメッセージは、教科書の各章の最後にあるので、よく見て対処していくこと
  - 各章の内容で、よく表示されそうなメッセージが掲載されている

プログラミングの考え方

プログラム作成の基本(1)

- プログラミングは、一種のパズル
  - 出された問題を内容ごとに分解
  - 各内容にあてはまる処理方法を検討
    - 授業の各回から、使えるものを探す

プログラム作成の基本(2)

(1) (2) (3) (4)

Ex. 標準入力でJavaのファイル名を入力し、拡張子を「.class」にして出力

(1): 標準入力

- 第6回授業の内容(入力)

(2): Javaのファイル名 → 文字列のデータ

- 第2回授業の内容(変数宣言)
- 第4回授業の内容(文字列のデータ型)

(3): 拡張子を変更 → 文字列のデータの操作

- 第4回授業の内容
  - 部分文字列の位置を探す操作(indexOfメソッド)
  - 文字列の一部分を変更(substringメソッドで部分文字列を取り出し、「+」で文字列を連結)

(4): 出力 = 標準出力

- 第6回授業の内容(出力)

プログラム作成の基本(3)

(1) (2) (3) (4)

Ex. 標準入力で2つの数を入力し、合計を計算

(1): 標準入力

➡ 第6回授業の内容(入力)

(2): 2つの数 ← (1)の標準入力で入力されるものは文字列!

➡
- 第2回授業の内容(変数宣言)
- 第6回授業の内容(文字列を数に変換)

(3): 合計を計算

➡ 第2回授業の内容(足し算)

(4): 明示されていないが、処理結果の確認には出力が必要

➡ 第6回授業の内容(出力)

プログラミングで必要なこと(1)

- 習ったことを整理して頭の中に入れておくこと
  - 変数はどのように扱うか?
    - 宣言・代入・参照・初期化
  - 文字列はどのように扱うか?
    - 文字列の連結・メソッドでの操作
  - 入力の時に行う処理はどんなものがあるか?
    - 文字列として入力データを受け取り・数値への変換

この程度の内容を頭に入れ、必要に応じて取り出せる(パズルができるようになる)ことが重要

- 処理の書式・テンプレートやメソッドの名前などは覚えなくても良い
- 必要なときに調べて使えるようになれば良い
  - ✓ 様々なプログラムを書いているとそのうち覚える

プログラミングで必要なこと(2)

- 1つ1つの命令の意味を考えること
  - この命令は何をするための命令か?
  - この命令の処理結果はどのようになるか?
- 常に「なぜ???」を考えること
  - なぜこの命令が必要か?
  - なぜこの命令をここに書かなければならないか?
  - なぜこの命令をここで使うことができるか/できないか?

常にこういうことを考えながら、エラーメッセージや例で示されている
プログラムを見ること

- 例を参考にプログラムを作るときに、各命令の要・不要・順序などを
  きちんと考えられるように

課題のお知らせ

- 課題内容と〆切:
  http://www.cis.twcu.ac.jp/~junko/Programming/Java1/
  - 質問は、大学のメールアドレスから送られたメールでなければ受け付けない