第１４話（Java 言語のポイント）

タヌキ、Java 言語を使えるようにする為には、クラス、インスタンス、コンストラクタという用語とその意味を理解することが絶対条件なのだ。でなければ、全体像が見えないまま部品をくっつけていく労働者になってしまうよ。

キツネ、クラス、インスタンス、コンストラクタが重要なのはわかったが、クラスって何だ？

クラスは、Ｃ言語の構造体のようなものだ。ただ、構造体はメンバ変数（データ）のみだったが、クラスはメンバ変数＋処理（関数）をパッケージにしているのだ。クラスを理解し易くする為に、第１３話でＣ言語を取り上げたのさ。

クラスは、構造体と同じで枠組みだけだからね。良く例えられるのは、タイヤキの鋳型だ。鋳型がクラス。この鋳型からは、餡入りもできるし、クリーム入りもジャム入りのタイヤキも作ることができる。できた各種タイヤキが実際に食べることのできるタイヤキとなるので、インスタンス（実体）とかオブジェクト（物）と言うのだ。タイヤキの鋳型は食べることができないので、クラスなのだ。

オブジェクトを作ることをインスタンス化ともいうよ。

クラスのイメージ図を書くと左図のようになる。

クラスのイメージ図

Student （クラス名）
string　namae ： （フィールド）
void　keisan( ) （メソッド（関数））

上図のクラスのイメージ図をJavaのプログラムで表わすと右のようになるよ。

```
class Student{
    String namae ;
    int kokugo;
    int sugaku;
    int goukei;

  void keisan(String simei,int ka2,int ka3){
    namae = simei;
    kokugo = ka2;
    sugaku = ka3;
    goukei = kokugo + sugaku;
   }
}
```

なるほどな。それで、このStudentクラスはどこに存在するのだ。

これは、実体の無い枠組みだけだからプログラムを格納する領域、
つまりメモリ上のコード領域だな。
次は、コンストラクタとインスタンスだね。
以下の代入式が重要だ。

Student　seito 1　＝　new　Student( ) ;

（クラス）　（参照型変数）　（new 演算子）（コンストラクタ）

右の関数（メソッド）Student( )がコンストラクタだ。

コンストラクタって、どこに作られる関数だ。ソースファイルの
どこにもコンストラクタの記述は無かったと思うが・・・・！

プログラマが定義しないコンストラクタは、ソースファイルをコンパイルする時に自動的に作られるのさ、このようなコンストラクタをデフォルトコンストラクタというのだ。当然プログラマが明示的に定義してもかまわないよ。コンパイルする時に自動的に作られるから、クラス名（Student）が、コンストラクタ名になるのだ。
new 演算子がコンストラクタ（Student( )）を呼び出して、インスタンス（実体）を作るのだ。ポインタ変数と同じなのだが、参照型変数（seito1）には、コンストラクタによって作られるインスタンスのアドレスが保存されるのだ。コンストラクタは引数無しで、インスタンスを作っておいて、後からメンバ変数にデータを代入しても良いし、最初から引数をセットしても良いんだよ。

キツネ、コンストラクタはコンパイル時に自動的に作られるのなら、実行ファイルに存在するのだろうから良いけどね、インスタンスは、seito1、seito2、seito3 といくつでも作れると思うんだが、メモリのどこの領域に作られるのだ？

タヌキ、良いことに気がついたな。インスタンスは、メモリのヒープ領域に作られるのだ。そのヒープ領域のインスタンス毎の先頭アドレスが参照型変数の seito1、seito2、seito3、の格納されるのさ。ただ、インスタンス内の関数（メソッド）の変数と値は、スタック領域に置かれるのだ。わかるかタヌキ！
もう少しわかるように以下に図示してみるか。

プログラム例（Rei.java　とする）

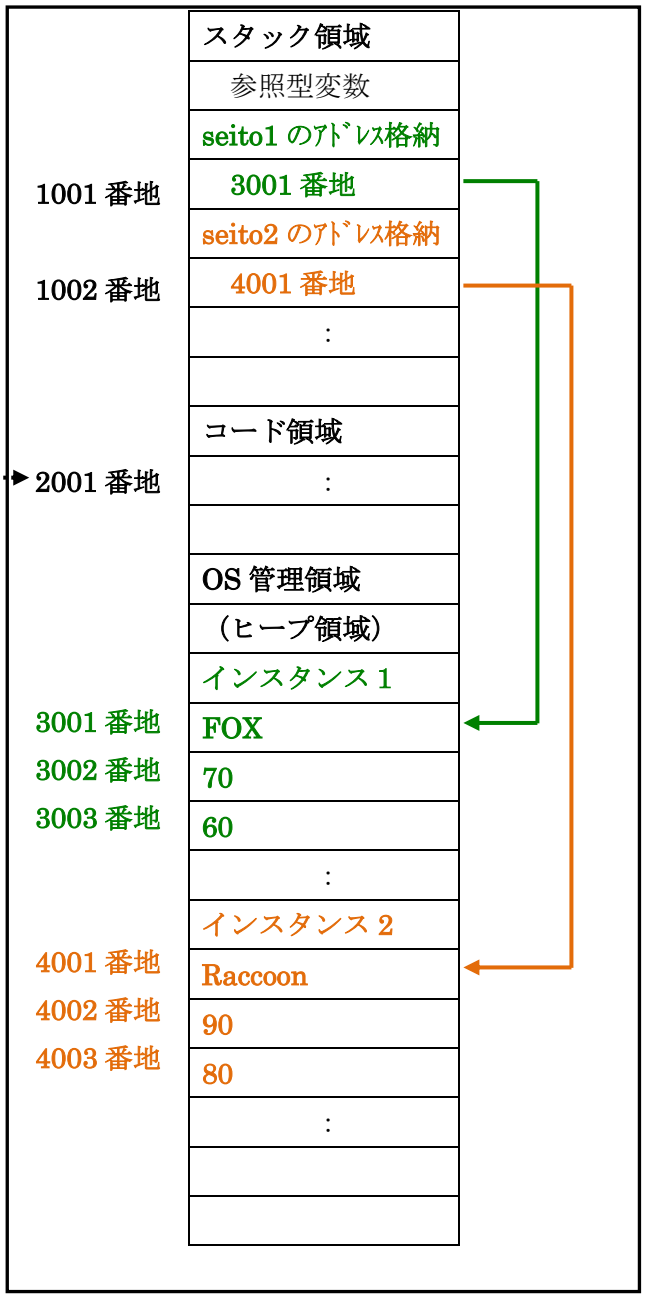
```
public class Rei{
   public static void main(String args[]){
      Student seito1   = new Student();
      seito1.keisan("FOX",70,60);
      seito1.kekka();
      Student seito2   = new Student();
      Seito2.keisan("Raccoon",90,80);
      Seito2.kekka();
    }
}

class Student{
    String namae ;
    int kokugo;
    int sugaku;
    int goukei;

   void keisan(String simei,int ka2,int ka3){
      namae = simei;
      kokugo = ka2;
      sugaku = ka3;
      goukei = kokugo + sugaku;
    }

  void kekka(){
   System.out.println("namae:" + namae);
   System.out.println("goukei:"+goukei);
   }
}
```

メモリの格納イメージ



Student クラス

namae
kokugo
sugaku
goukei
keisan()
kekka()

Java 言語は、プログラム（Rai.java）だけを見ていてもなかなか理解しずらいよ。右側のメモリの格納イメージと組み合わせて考えると理解が進むかもしれない。 プログラムの２行目に main( ) 関数があるが、C 言語と同じでプログラムの中で最初に実行される関数なのだ。３行目にオイラ（seito1）の実体（インスタンス）を作ってくれている。オイラの実体が［3001 番地］にあるよ、と言ってくれているのが、参照型変数の seito1 だよ。参照型変数というのは、C言語のポインタ変数と同じだよ。注意してほしいのは、seito1 は、スタック領域に作られ、オイラの実体（インスタンス）は、ヒープ領域に作られる点だ。
ヒープ領域は、ＯＳ管理領域に属しているのだ。
new 演算子で呼び出されるコンストラクタ（Student( )）は、12 行目以降に定義されている Student クラスを睨みながらオイラの実体を作ってくれているのさ。
同様に６行目にタヌキの実体を作って、その所在地を seito2 に格納しているぞ。
右下の Student クラスは、レイアウトだけなので、プログラムを格納するコード領域に格納されているのだ。

キツネ、メモリ領域を図示してくれたので、なんとなくわかったような気がする。でもスタック領域とかヒープ領域とか、よくわかんないな！

スタック領域は、確保されるメモリ容量が固定されているのだ。主に、メインプログラムからサブプログラムに制御を移した時に、メインへの戻り先（アドレス）を格納するのに使われている。情報の試験でも PUSH（取り出し）、POP（格納）の用語とともにスタック領域を勉強することになるよ。
ヒープ領域は、プログラマが必要なメモリ領域を明示的に確保でき、使用し終わったら解放することもできる領域なのだ。

なるほどな！勉強を深めれば深める程、次に何を勉強しなければならないか見えてくるのか。だから、勉強すればするほど知らないことが次々にでてきてオイラは何にも知らないのだ、と謙虚な気持ちになるのか。
オイラ、PUSH、POP を勉強した時に、データを入れたり、出したりする所、「先入れ後だし法」と教えられたけれど、プログラムとの繋がりがわかっていなかった。情報の試験では、解答できたけどね。

タヌキ、成長したな！
さらに、スタック領域は確保されるメモリの量が固定され、小さいので、格納された戻り先アドレスを取り出さないで、サブプログラムにどんどん入って行くと「OUT　OF　MEMORY」（メモリ領域のパンク）というエラーメッセージが出てプログラムがフリーズするよ。そのような時に、実行しているプログラムは小さく、メモリーは２GB も積んでいるのに、なんでフリーズするの、と悩むことになるのだ。
ちょっと話がズレたかもね。
しつこいけれど、もう一度繰り返し言っておくね。

Java 言語のポイントは、

クラス、インスタンス、コンストラクタ

を理解することです。

第１５話は、Java 言語の特徴の話でもするか。