情報科学実験１　　第０回　総復習　（プログラムの作り方）　　furuya

step1:額縁を作る

```
public class Abc {
        public static void main(String[] args){
            “プログラム”をここに書く
        }
}
```

Abc はプログラム名なので、自分で適当に決める

“プログラム”は、次の３つの部分からなる。

- 変数宣言
- 変数への初期値代入
- プログラム本体

step2:　　変数宣言

先ず、プログラムで使う変数を決めて宣言する。

（int a, b, c;なんてやつ）

変数名のつけ方：プログラムが大きくなると変数を沢山使うので、どれが何だったか分からなくなる。そこで、自分で覚えやすい名前をつける。但し Java には“予約語”という、勝手に使ってはいけない名前がある。それ以外なら何でもよい。予約語は、付録１につけます。また、この資料では（皆さんが定義する）変数をローマ字表記します。ですから、ローマ字表記してある変数は i や j や k に書き換えても何ら問題ありません。

変数宣言時に初期値を代入できます（int a=5;なんて初期値を代入しちゃってもよい）。

＜配列＞分かっていない人は付録２参照

配列を使う時は配列も宣言する。配列とはデータが表形式で用意されている時使うもので、縦横の枠番号でデータを指定する。

3	2	5
7	1	4

右の２次元配列（配列名 hairet だとする）を宣言するには、int hairet[][]={{3,2,5},{7,1,4}};であり、プログラム中で hairet[0][2]と書くと、その変数の値は５となる。

step3:　変数への初期値設定（代入）

a=3; b=6;なんて書く。=は数学の式の=じゃなくて、<=（左向き矢印）即ち数値を書き込むことを示している。Pascal という言語では＝じゃなくて:=って書いて書き込む方向を示す。だから、a=b+c;はあっても a+c=c;なんてことはありえない。 a=a+1;は a に a+1 を代入、ＯＫだよね。

右図のように宣言 int　hensu1=5;

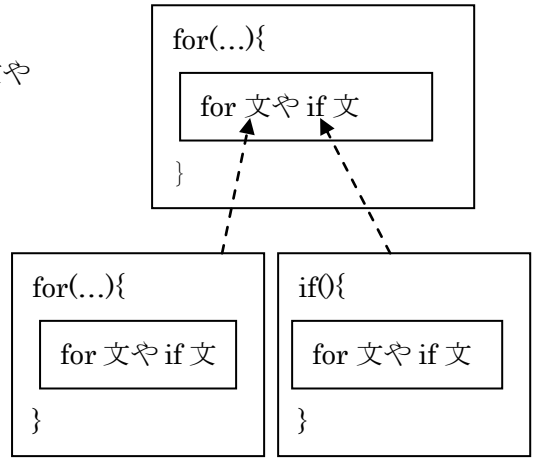
で hensu1 の初期値を５にセットしても（いつも固定ではなく）、プログラム中で、hensu1=3;と書けば hensu1 には３が上書きされ、hensu1=hensu1+4;とあれば、hensu1(既に３となっている)に 4 を足した結果 7 が hensu1 に上書き(代入)されるんだ。

```
int hensu1=5;

hensu1=3;
hensu1= hensu1+4;
```

配列への初期値の代入は繰り返し（for 文）を使う。
例えば上の２次元配列の要素に全部５を
入れたければ、右のようにして５を代入する。

```
for(int tate=0;tate<2;tate++)
   for(int yoko=0; yoko<3;yoko++)
      hairet[tate][yoko]=5;
```

step4: プログラム本体の設計
基本的にプログラムは、“繰返し文”の中に if 文や
繰返し文が入れ子に入っている。
入れ子とは、ロシヤの人形で中に又同じような
人形が入っている感じ。右図参照。



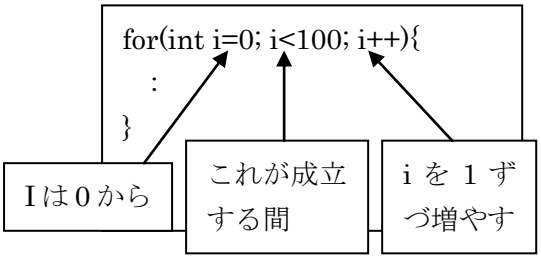
繰返し文には、
for（繰り返し回数が分かっている時使う）と
while（繰り返し回数が分かっていない時使う）
があるが、初心者は for 文を使うこと。

＜for 文の復習＞
右のプログラム最初のループは

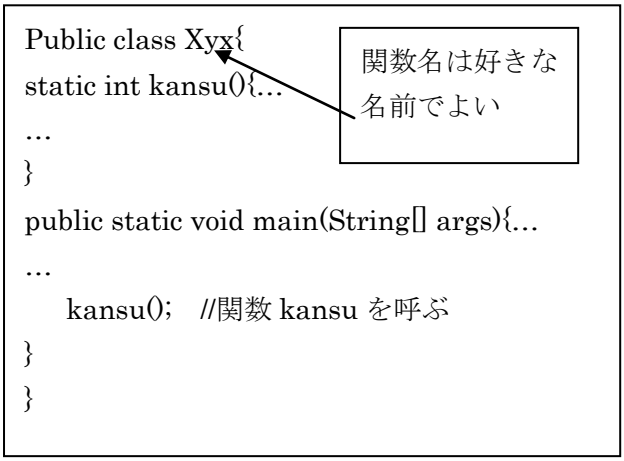
```
int a=0;
for(int i=0;i<2;i++)
   for(int j=0; j<3;j++)
      a= a+i+j;
```

i=0　j=0 なので、a は a<=0+0+0 で、a は 0、次は
i=0,　j=1 なので a は a<=0+0+1 で、a は 1、次は
i=0,　j=2 でなので a は　a<=1+0+2 で、a は 3、次は
i=1　j=0 で a は a<=3+1+0 で a は 4、次は
i=1,　j=1 で a は a<=4+1+1 で a は 6、次は
i=1,　j=2 で a は　a<=6+1+2 で a は 9、となる。
蛇足：
１．右のプログラムでは for(int i=0; i<2;i++)と i を for の中で宣言している。もしプログラムの最初の方で int　i;と宣言しておけば、for(i=0; i<2; i++)でよいが、Java では for の中で宣言する人が多い（理由は省略）。

２．　100 回繰り返す場合の基本形は
右の形ですが、別に i を 0 から始めなくても
かまいませんし、i を２つずつ増やしたけ
れば i++でなく、i=i+2 とすればＯＫです。
状況に応じて決めてください。



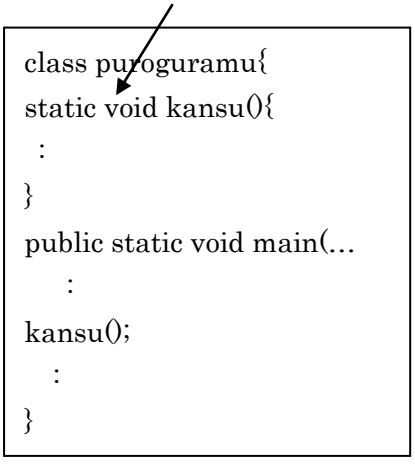
関数（メソッド）：(右図参照)
step1 のプログラム構造と似ているが main(){…}の前や後に、kansu(){…}の様なものが入っているプログラムもある。
これは関数（Java ではメソッド）と呼ばれ、main()の中で何度も使われるルーチン（プログラムの一部）に名前を付けて、外に掃き出したものです。



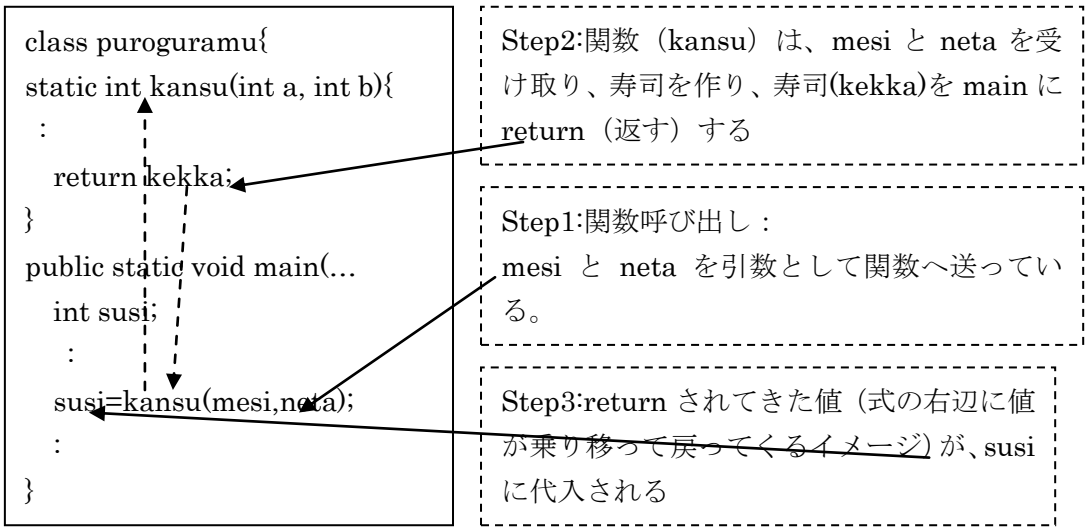
（別の言い方をすると）
プログラム中でよく使う部分を予め main の外に作っておいて、main の中ではそれを略した名前で使用する、ってこと。
（プログラムの流れ）
main の最初から命令を実行していき、関数 kansu に到達すると、kansu()の方に飛び kansu の本体を実行し、kansu の最後の命令に到達すると main に戻り kansu()の次の命令から main の実行を再開する。

（(ヒント）関数は一種の専門家で、main から何度でも呼ぶことができる。呼ぶとは、関数に飛んで処理を行い（普通）計算結果を main に返してくることです。
計算結果を main()に返す命令が return 文です。また返す値の型を関数名の前に書いておく必要がある。）

関数は大きく分けると２種類ある。
第一は関数の中で仕事が終わってしまい、main に結果を返さないタイプ（廃棄物処理業者みたいなやつ)。このタイプの関数を宣言する時は、関数名の前に static void を付ける(下図)



第２のタイプは結果を main に返すタイプで、“料理人”（飯とネタを“引数”としてもらい寿司を返（return）してくる)って感じ。
この場合は、関数宣言時に関数名の前に main に返す値の型（例えば int）を付ける。また、main に結果（値）を返すには return 命令を使う。
（注：return は関数の先頭に戻る、という意味ではなく、main に結果を返す、って意味）（次図参照）



情報科学実験１　Ｂクラス　第１、２回　課題

１．次の総和を求める　(1)0~99,　(2)1~100,　(3)1,3,5,…99 奇数

（HINT：総和を入れる変数(souwa にしよう)が必要なので宣言し、最初は 0 だな、と思う。プログラム本体は繰り返し文だなって思うので for(int i=0;i<100;i++){…}を用意する。{} の中は i を足していけばいいや、って思って完成。）

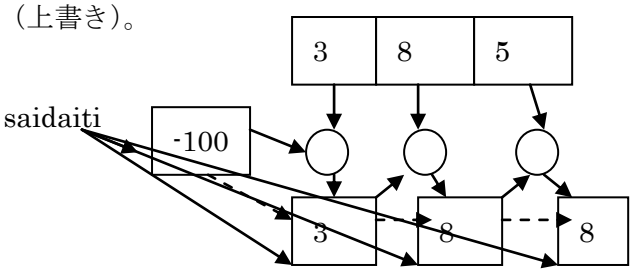
２．１次元整数配列 a（要素数５）に、左から１，２，３，４，５という数値を代入せよ（for 文の中で代入文使用）。その後、a の中身を出力せよ（for 文）。

(HINT : 配列 a を宣言し、初期値を for で代入、a の中身を、端から出力するのは for 文だよな、って考える。)

３．１次元整数配列 a（要素数５）に、左から８，７，６，５，４という数値を代入せよ（for 文使用）。その後、a の中身を出力せよ（for 文使用）。

４．１次元整数配列 a（要素数５）に適当な数を入れておく。配列 a の中の最大値を出力せよ（for 文と if 文を使用）。

(HINT : 最大値を求めるには、saidaiti とか（どんな名でもいいんだけど）いう変数を用意し、これに小さな値（例えば-1000）を入れておき、for で配列要素と比較して要素の方が大きかったら saidaiti に入れる（上書き）。



５．１次元整数配列 a（要素数５）に適当な数を入れておく。配列 a の中の最小値の入っている要素番号を出力せよ（for 文と if 文を使用)。

1	2	3
11	12	13

６．２×３の２次元整数配列に右図のように for 文と代入文を使って数値を入れ、配列完成後その中身を配列通り２段出力せよ。

７．２×３の２次元配列に適当な数をいれておく。各列で数値を比較し、小さいほうの値を上（行０）になるよう入れ替えろ。
８．２×３の２次元配列 a に適当な数をいれておく。各列で数値を比較し、大きいほうの値を１次元配列 dai の対応する場所に入れる。
９．２×３の２次元配列に適当な数をいれておく。各行の最大値を求めよ。
１０．ookii メッソド（関数）＜ookii(a,b)：a,b の大きい方を返す＞を作る
１１．べき乗を求める関数＜bekijo(a,b)：a の b 乗値を返してくる＞を作る
(HINT：べき乗は 5 の３乗とか（一般に）２乗より大きいやつ。５の３乗なら１×５×５×５だから結果を入れる変数を決めて（例えば bekijo とか）宣言し、５を３回 for ループで掛ける。)

１２．bekijo を使って、２次元配列 a の列ごとに、a[0][i]の a[1][i]乗を b[i]に格納
１３．Java には Math.random()という関数が用意されていて、0~1 の間のデタラメな実数を返してくる（乱数)。これを使って、０～９の整数乱数を返す関数を作れ。
１４．キーボードから 1,2,3 のいずれかの数値を入力すると、1 なら Jan,2 なら Feb、3 なら Mar と出力するようにせよ（switch case 文使用）

(ヒント)キーボードからの読み込み：

```
import   java.io.*;
:
int   suuti;
BufferedReader br= new BufferdReader(new InputStreamReader(System.in));
suuti= Integer.parseInt(br.readLine());
```

```
switch case  文
switch(変数名) {
      case(値):   処理; break;
       :
}
```

お薦め CPad for J2SDK(プログラム開発がグッと楽に)
http://www.vector.co.jp/soft/win95/prog/se153698.html
(注）インストール時に javac の在かを聞いてくるので、C:¥j2sdk……¥bin¥javac.exe と書く必要があった(WinXP の場合)。上の……はバージョンにより異なる。

付録１：Java 予約語（プログラム中で変数名に使ってはいけない単語）

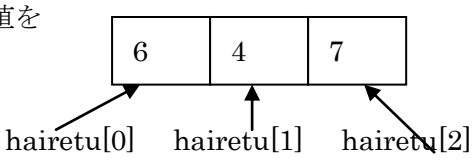
abstract,boolean,break,byte,case,catch,char,class,const,continue,default,do,double,else,extends,false,final,finally,float,for,goto,if,implements,import,instanceof,int,interface,long,native,new,null,package,private,protected,public,return,short,static,super,switch,synchronized,this,throw,throws,transient,true,try,void,volatile,while.以上

付録２：配列（変数を並べたもので、配列名と位置で変数を指定します）

（１） １次元配列：右図は配列名 hairetu（配列名も変数名と同じで自分の覚えやすい名前を付ける)。配列要素（数値の入るマス）数３。

この配列を宣言して、ついでに右のように初期値を入れる（代入するという）には次のようにする。

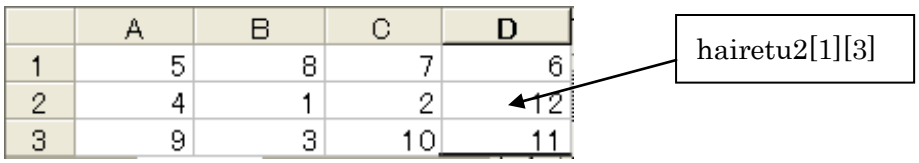
int　hairetu[]={6,4,7};



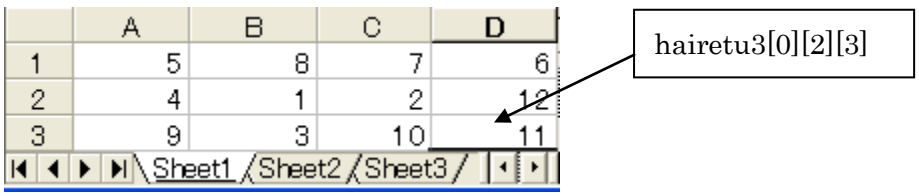
（２） ２次元配列

下図はＥＸＣＥＬの表の一部です、２次元配列は、この表のイメージです。これは３×４の２次元配列です。EXCEL ではセル（マス）を（３．Ａ）なんて指定します（３．Ａは９)。しかし Java では下図の縦 1,2,3,を０，１，２と指定し、横 A,B,C,D も０，１，２，３と指定します。ですから、もし、下の２次元配列を hairetu2 とすると、hairetu[2][0] の値が９になります。要素位置指定は“縦”“横”の順ですので注意してください。この２次元配列を宣言し下のように初期値をセットするには次のようにします。

int　hairetu2[ ][ ]={{5,8,7,6},{4,1,2,12},{9,3,10,11}};



（３） ３次元配列：下図は EXECL の表です。今度はシート番号も指定できます。これが３次元配列です。セル（マス）３．Ａ．Sheat1 が９です。Java では Sheat1 を０、Sheat2 を１、Sheat2 を３と指定しますので、Sheat1 の３．B は、hairetu3[0][2][1] となります。位置指定が“奥行き（Sheat)”、“縦”、“横”、の順ですので間違いなきよう。



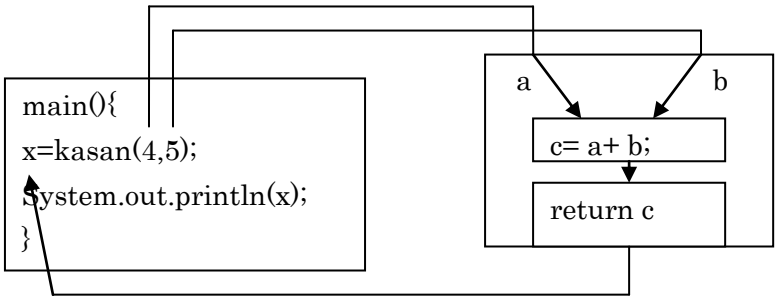
付録３　if 文の例

```
if((tebnsu>=60)&&(shuseki>8)) seisaki=1;
else seiseki=0;
```

情報科学実験Ｉ　Ａクラス第１回（Java の真髄、オブジェクトへの道標）

お薦め CPad for J2SDK(プログラム開発がグッと楽に)
http://www.vector.co.jp/soft/win95/prog/se153698.html （注）インストール時に javac の在かを聞いてくるので、C:¥j2sdk……¥bin¥javac.exe と書く必要があった(WinXP の場合)。上の……はバージョンにより異なる。

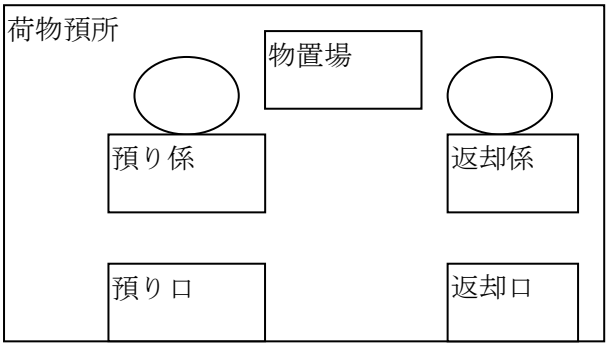
（１） 関数は専門家（寿司屋）だって話があった。
ただ、この専門家は材料をもらって寿司をつくることはできるが、材料を保存したりはできない。簡単に言えば、入力を計算し結果を垂れ流す専門家だ。



それじゃ、“荷物一時預り”のような専門家は作れない（だって関数が前に呼ばれた時の結果などをしまって置けないんだから)。

（２） これを解決するため Java では、“オブジェクト”、と“クラス”を導入した
オブジェクト＝倉庫を持った専門家集団（冷蔵庫を持った、“すし職人”と“お茶出し”がチームを組んだもの、ってとこ)。
クラス＝オブジェクト派遣会社（(専門家の仕事＋倉庫）＝オブジェクト）がプログラムとしてここに書いてある）
例）荷物を１つしか預かれない“荷物一時預り専門家集団”オブジェクトを考える
オブジェクト＝｛荷物置き場：倉庫（フィールド）に対応；
　　預かり専門員：専門家１（メソッド）に対応；
　　荷物返却専門員：専門家２(メソッド)に対応；｝

（注）専門家集団（チーム）の各専門家は一つの機能しか担当しない。

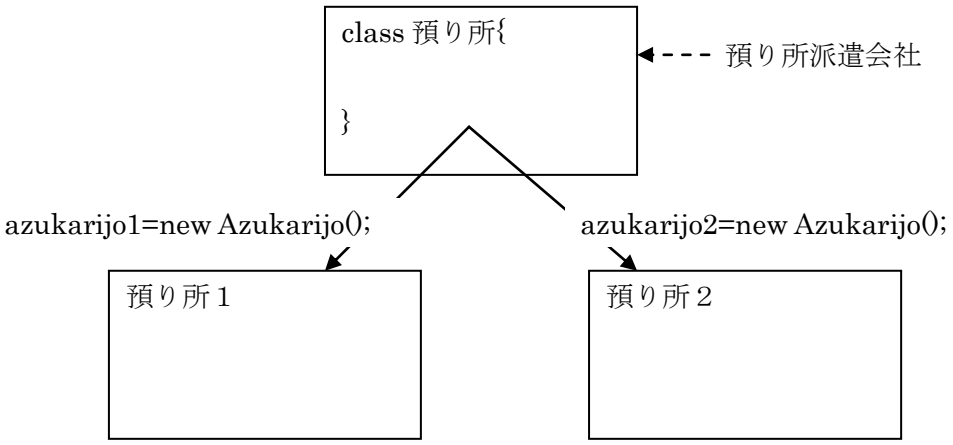


プログラムにすると、

```
class Azukarigumi{
    int   okiba;
    void azukarigakari(int ni){
        okiba= ni;
    }
    int henkyakugakari(){
```

```
        return okiba;
    }
}
```

これは class（クラス）ですので、“荷物預りチーム”派遣会社です。
main()の中では、新しい“専門家チーム派遣依頼”を new で行い、チームには main の方で名前を付けます。
Azukarigumi azukarijo1=new Azukarigumi();
Azukarigumi azukarijo2=new Azukarigumi();
で２つのチーム（預り所）が派遣されました。
預り所１へ荷物(5)を預けるには、azukarijo1.azukarigakari(5);
預り所１から返してもらうには、modorini=azukarijo1.henkyakugakari();とする。



例２　年齢を答える犬
年齢を答える犬（２匹）を作る。先ず、Dog という class を作る。Dog は、コンストラクタ、age というフィールド（変数)、howold というメソッドからなる。コンストラクタは年齢を引数で受け取り、age に代入する。メソッド howold は age を出力する。
次に　I というメイン（main）class を作る。main では２匹の Dog（pochi３歳と koro４歳）を作り、これらの犬に howold と質問する。
＜解法の手引き＞
専門家チームは“年齢を答える犬”である。犬派遣会社(class Dog)には犬の機能がプログラムされている。
専門家（犬）の倉庫には年齢を書いた紙が入っている。
専門家チームには、年齢を答える専門家犬 howold が１匹いるだけである。

```
class Dog {
    int age;
```

```
    Dog(int a){
        age= a;
    }

    void howold(){
            System.out.println(age);
    }
}

public class I {
 public static void main(String[] args){
        Dog pochi= new Dog(5);
        Dog koro= new Dog(3);
        pochi.howold();
        koro.howold();
 }
}
```

(注)Dog 内の Dog(int a){….};っての（クラス名と同じ名のメソッド）はコンストラクタと呼ばれ、main()の new Dog(3);で新しい犬を作る（派遣してもらう）時呼ばれ、フィールドの初期値設定に使われる。コンストラクタで何もする必要がない時はコンストラクタを書かなくてよい。

課題：

１．ボタンを押すと警戒音が鳴る防犯ブザーを３名に配布してボタンを押してもらおう。

２．吠えろと言うと吠える犬種 Hoeken がいる（として）、Hoeken を２匹作り吠えさせてみよう。

３．ネットワークカードには IP アドレスが割当てられていて、ipconfig と聞くと IP アドレスを返してくる。ネットワークカードを３枚(eth0,eth1,eth2)作り、各カードの IP アドレスを聞いて確認しよう。IP アドレスは constructor で指定する。ここでの IP アドレスは、
192.168.0.～の～の数値とし、適当に決める。

４．nanten と聞くと成績を返してくる個人別成績カードがある。３人の成績カードを作り、点数を聞いてみよう。点数はコンストラクタで指定。

５．ラジコンカーがある。操縦器(プロポ)には３つのボタン（前進、右進、左進）のボタンがあり、それを押すと１ｍ進み、現在位置（ｘ、ｙ座標）と車番を出力する。２台のラジコンカーを作り、走らせてみよう。車番はコンストラクタで指定。

６．今どこを走っているかを知らせながら走る車がある（位置と車番を出力）。車は move()

という指示で 1km づつ進む。車を３台作り、走らせてみよう。

補足１：今までは main クラス内で、外部クラスのオブジェクトを生成して使っていました。これから GUI や複雑なプログラムを作る場合、main クラスのオブジェクトを作るケースも増えてきます。例をあげておきますので、よく見て慣れておきましょう。

例１：私の飼っているポチに餌をやる

```
class Dog{
	void eat(){	//餌を食べる
		System.out.println(“gochisousama”);
	}
}
class Owner{
	Dog pochi= new Dog();
	void esayari() {
		pochi.eat();
	}
	public static void main(String[] args){
		Owner watasi= new Owner();
		watasi.esayari(); //餌をやる
	}
}
```

例２：私の車を運転する

```
class Car{
	int speed=0;
	void accel(){
		speed++;
	}
	void brake(){
		speed--;;
	}
}
class Driver{
	Car mycar;
	Driver(){		//constractor で車作り
		mycar= new Car();
	}
	void drive(){
		mycar.accel();
		mycar.brake();
	}
	public static void main(String[] args){
		Driver watasi= new Driver();
		watasi.drive();
	}
}
```

補足２：“static って何だ？”

Java のアプリケーションプログラム：オブジェクト（専門家チーム）を敷地(メモリ)内に、必要に応じて、適切な場所に（new で）派遣し（作り）、処理を行う。でも、プログラム実行の一番最初には、オブジェクトを作るオブジェクトが無いと何もスタートできないの

で１つだけ最初から敷地（メモリ）の決まった（プログラムが終了するまで変わらない）場所に割り当てられるオブジェクトがあります。それが main(){〜} の、〜の部分のオブジェクトです。そして main()の前に付いている static はメモリの決まった位置、ということを示しています。main のクラスで使う変数は static（メモリ内固定位置）になければいけないので、main()のクラスでは static の付いた変数を使っていました。(static は国有地にあるものって感じ)

皆さんが今までやってきたプログラムはほとんど main クラス以外のクラスを使っていない特殊なプログラムでした。これからは、自作のクラス（前回の Dog とか Azukarijo）を作っていくことになります。その時、それらのクラスから作られるオブジェクトはメモリ内に必要に応じて必要な場所に置かれるべきものですので、main クラス以外では static を使わないようにしましょう。

２）前回のクラス２例の機能対応（並べると対応が分かりやすいと思い並べてみました）

```
Class Dog{
   int age;
   howold(){
   System.out.println(age);}
}
class Owner{
   main(){
     Dog pochi=new Dog();
     Dog kuro= new Dog();
     pochi.howold();
     kuro.howold();
   }
}
```

```
class Azukarijo{
   int okiba;
   azukeru(int nimotu){
      okiba=nimotu;}
   henkyakui(){
        return okiba;}
}
class Owner{
  main(){
   Azukarijo mkimae=new Azukarijo();
   Azukarijo depatika=new Azukarijo();
   ekimae.azukeru(hon);
   depatika.azukeru(tabemono);
   te=Ekimae.henkyaku();
   }
}
```

情報科学実験Ｉ　第２回　継承、抽象クラス、インタフェース

Ｉ、継承

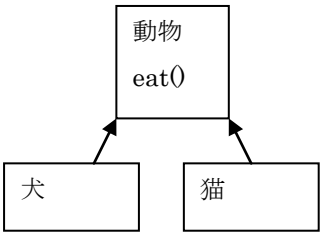
Act1:　継承とは、上位クラス（スーパークラス）の機能（メソッド等）を下位クラス（サブクラス）から使える、というものです。構文は以下の通りです。

class サブクラス名 extends スーパークラス名｛クラス本体｝

Act2:　プログラム例２－１

犬、猫は動物である。飼い主が、ポチとタマに“食べろ”と言うと、動物の eat()が動いて、動物の本能が“満腹”と言ってきます。

```
class Animal {
        boolean isHungry= true;
        void eat() {
                isHungry= false;
                System.out.println(“manpuku”);
        }
}
class Dog extends Animal{  }
class Cat extends Animal{  }
class Owner {
        public static void main(String[] args) {
        Dog pochi= new Dog();
        Cat tama= new Cat();
        poch.eat();
        tama.eat();
}
```



ＩＩ、Act3：抽象（abstract）クラス

これは Java の特徴であるポリモフィズムを実現する方法の１つです（もう一つは後で説明するインタフェースです)。ポリモフィズムは多様性とも呼ばれますが、簡単に言うと例えば次のようなことです。

（１）　社員に“働け”と言うと、プログラマはプログラムを、事務員は事務をすること。

（２）　ペットに“鳴け”と言うと、犬はワン、猫はニャーと言うこと。

abstract クラスでは、メソッド名だけを明記しておき、下位のクラスで独自のメソッドを書き込みます。

Act4：プログラム例２－２：犬、猫は共に動物で、餌を食べます（eat()）。しかし、犬はワンと、猫はニャーと鳴きます。２匹は動物を継承（extends）しており、別の籠に入れ

ている。犬と猫に“鳴け”と言って（main()の中で)、ワン、ニャーと鳴かせなさい。

```
abstract class Animal {
        void eat() {
                System.out.println(“manpuku”);
        }
        abstract void cry();
}
class Dog extends Animal{
        void cry() { System.out.println(“Wan”);}
          }
class Cat extends Animal{
        void cry() { System.out.println(“Nyaa”);}
          }
  }
class Owner {
        public static void main(String[] args) {
                Dog pochi= new Dog();
                Cat tama= new Cat();
                pochi.eat();
                pochi.cry();
                tama.eat();
                tama.cry();
}
```

以上、abstract の話でした。

ポリモフィズムは、Owner の部分が少し変わります。

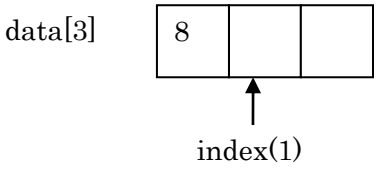
```
abstract class Animal {
        void eat() {
                System.out.println("manpuku");
        }
        abstract void cry();
}
class Dog extends Animal{
void cry() { System.out.println("Wan");}
  }
class Cat extends Animal{
void cry() { System.out.println("Nyaa");}
  }

public class Owner {
    static Animal [] pets= new Animal[]{ new Dog(), new Cat()};
    public static void main(String[] args){
        pets[0].cry();
        pets[1].cry();
    }
}
```

注目：pets は Animal クラスで、pets[0].cry()は猫と犬に鳴けと言っているのではなく、動物に鳴けと言っている点です。動物は鳴くものだ、という前提で、鳴けと言っています。

（念を入れたい人へ）abstract の例を、もう一つ位見たい人は付録A参照

Act5:

課題２－１　FIFO と LIFO を作れ。これらは共に int data[3]の配列を用意し、put(…)でデータを書き込み、get()でデータを取り出す（オーバフローと、空からの get()はないように使う)。



FIFO,LIFO 共 put(…)では配列の左(data[0])から詰めていく。int index は put(…)でデータを書き込む位置を指定する（最初は０、右図はデータが１つ書き込まれている状態)。

get()の場合、FIFO と LIFO で処理が異なる。

LIFO では index-1 の指す配列の中身を返し、

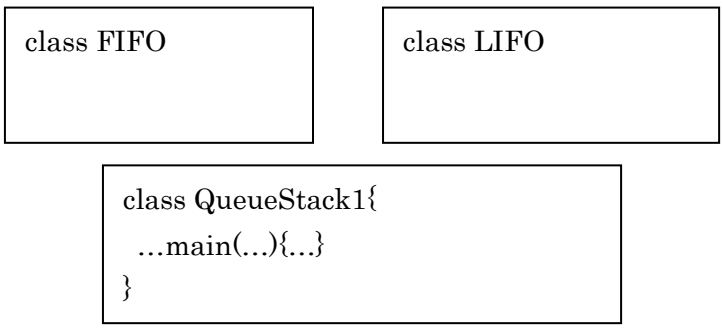
FIFO では data[0]を取り出し、左端が空になるので配列内を左にシフトする。

（FIFO 及び LIFO は別のクラスにし、get()と put(…)をメソッドにする）

main()では、get()で取り出した中身を出力するようにし、次の命令を実行する。

fifo.put(6); fifo.put(7);fifo.put(8);　fifo.get();fifo.get(); fifo.get();

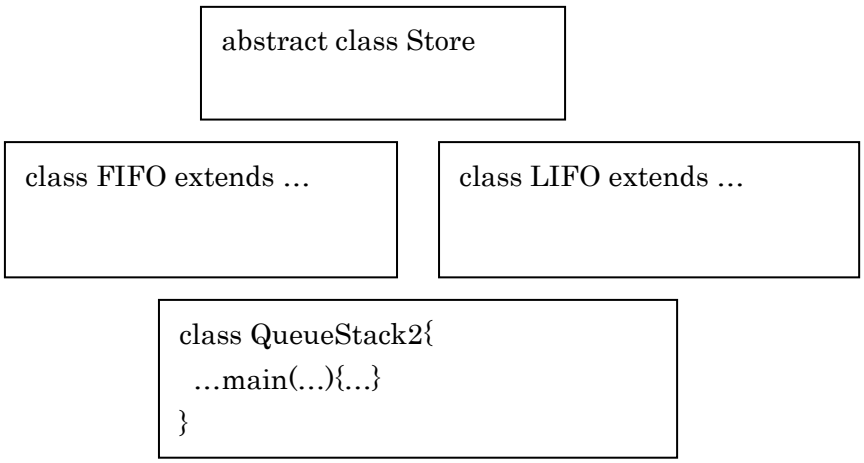
lifo.put(1); lifo.put(2);lifo.put(3);　lifo.get();lifo.get(); lifo.get();



プログラムの構造

課題２－２　FIFO と LIFO の共通部を Store というクラスにし、それを継承して FIFO と LIFO を作り、前課題と同じテストを行え。

（ヒント）data[]には put()と get()でしかアクセスできないようにしたい。data[]と put()は、FIFO,LIFO 共通である。そこで、put()は Store 内に置き extends で継承し、get()を abstract にして FIFO と LIFO 内で処理を記述する。



プログラムの構造

ＩＩＩ、インタフェース

（第４回以降のための準備：よく分からなくてよいので雰囲気だけ味わっておいてください）

第４回以降のプログラムにはクラス名の後に implements という言葉がついたものが使われます。これはインタフェースと呼ばれ、継承、抽象と深い関係があります。

クラスの宣言はこんな（次の）感じではじまります。

class　クラス名　implements　インタフェース名

初心者が出会うインタフェースは次の３つです。

（１） ActionListener インタフェース：第４回 Graphical User Interface で説明

プログラム構造次のような形で、画面上のボタン等を押されたら actionPerformed()に飛ぶので、この部分（イベント発生時の処理）は自分で書いてください、という意味。

```
public class Rei_1 extends Frame implements ActionListener {
    public static void main(String argv[]) {
        :
    }
    public void actionPerformed(ActionEvent evt) {
                :
    }
}
```

（２） MouseListener インタフェース：第５回 Applet で説明

プログラムの構造は以下のようで、マウスボタンを押した時の処理は太字部のメソッドに自分で書いてください、という意味。

```
public class Rei_2 extends Applet implements MouseListener{
        int x=-10, y=-10;
        public void init(){
            addMouseListener(this);
        }
        public void mousePressed(MouseEvent e){} //これらは使わなくても書く
        public void mouseClicked(MouseEvent e){}//
        public void mouseEntered(MouseEvent e){}//
        public void mouseExited(MouseEvent e){}//
        public void mouseReleased(MouseEvent e){}//
}
```

（３） Runnable インタフェース：第６回 Applet でアニメーションで説明
プログラムは以下のような形なので、run()の所は自分で書いてね、という意味。

```
public class Rei_3 extends Applet implements Runnable {
    public void start() {
            :
    }
    public void run() {
            :
     }
}
```

一寸見には､抽象と縁もゆかりもなさそうなインタフェースですが､深い関係があります。
上記（１）（２）（３）に共通するのは、特定のメソッドを“自分で書いてね“というものでした。
思い出すと、abstract では、上位クラスではメソッド名を決めておき、下位クラスで処理（泣き声）を書きました。
implements の“自分で書く”は、abstrct の下位クラスで“処理を書く”に対応しています。
関係を知りたい人は付録Ｃを読んでください。

付録Ａ　　abstract を使ったプログラム例
ビールとワインの価格に占める税の割合を計算するプログラム。下位から上位の変数が見えている点も注意、Tax から calculate の引数として価格を送っても面白い。

```
abstract class Alcohol {
     int price = 100;
     void show(int taxprice) {
         System.out.println(taxprice + "円です。");
```

```
    }
    //メソッド名を提示
    abstract void calculate( );
}

class Beer extends Alcohol {
    //抽象クラスで宣言されたcalculateメソッドの実装（オーバーライド）
    void calculate( ){
        show((int)(price * 0.465));
    }
}

class Wine extends Alcohol{
    //抽象クラスで宣言されたcalculateメソッドの実装（オーバーライド）
    void calculate( ){
        show((int)(price * 0.103));
    }
}
class Tax {
    public static void main(String[ ] args) {
        Beer superdry= new Beer( );
        Wine madonna= new Wine( );
        superdry.calculate( );
        madonna.calculate( );
    }
}
```

これも、上の動物の例にならい酒類の棚箱（配列）に色々な酒を入れておきポリモフィズムを実験してみよう。

付録Ｂ　基本情報の過去問に挑戦しよう。（平成１３年秋の問８）

http://www.rs.kagu.tus.ac.jp/infoserv/j-siken/H13b2/pm08.html

（ヒント）(知っておく価値のある例)
コンストラクタに注目してプログラムを見てください。　java Sub xyz と入力すると xyz と出力されます。subObject 生成時に Sub のコンストラクタが呼ばれ、その中で上位のコンストラクタを呼んでいます（super(s)）。これにより読み込まれた文字列が上位へ送られます。

```
class Superc {
    String cstring;
    //スーパークラスのコンストラクタ
    Superc (String s) {
        cstring = s;
    }
    //下位から使われるメソッド
    void sMethod( ) {
        System.out.println(cstring);
    }
}

public class Sub extends Superc {
    Sub (String s) {
        super(s); //上位クラス(Superc)のコンストラクタを呼ぶ
    }
```

```
    public static void main(String[ ] args) {
        Sub subObject = new Sub (args[0]);
        subObject.sMethod( );
    }
}
```

付録Ｃ：　ポリモフィズム第２弾、インタフェース（implements 実装）

抽象クラスではメソッド名を決めておいて、下位クラスでメソッドの中身を定義するという話がありました。しかし java は多重継承（extends）をできません。多重継承とは下の図のように、牛が食材と動物（複数のクラス）を継承しているケースを言います。要するに、extends A,B とは書けません。しかし、実際のプログラムでは多重継承を行いたい場合も発生します。この時には“インタフェース”というものを使います。インタフェースでは抽象クラス同様メソッド名を決めておき別のクラスで､メソッドの中身を定義します。

抽象クラスとの違い：

（１） インタフェースは複数指定でき、多重継承に相当する処理ができる

（２） 抽象クラスでは具体的（中身のある）メソッドを定義できるがインタフェースではできない

インタフェースの構造は次の通りです。

public　interface　インタフェース名　{抽象メソッド；..；抽象メソッド；}

インタフェースの実装は implements を使用して行います｡インタフェースは１つのクラスに対し、複数実装することもできます。

class　クラス名　implements　インタフェース名,・・・

また、クラスが extends を使用する場合は、implements は extends の後に指定します。

c lass クラス名 extend スーパークラス名　implements　インタフェース名

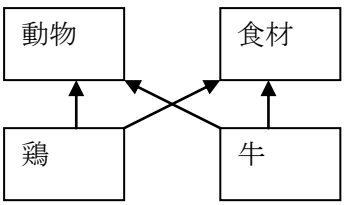
ポリモフィズムは一寸見方を変えると、“　implements　xxx　“となっている場合、

(a) xxx のメソッドを自分で“書きます”ということを言っている。とも言えるし、

(b) xxx が同じものは仲間で、共通の名のメソッドを持っている、とも言える。

Act2：プログラム例：鶏と牛がいる。共に鳴き方や料理法が異なる。鳴き方と料理法は別のクラスの概念なので、別のクラスにしたい。しかし、Java は複数のクラスを継承（extends）できない。それでも、２匹に鳴けと言い、代表的料理法を聞きたい場合はインタフェースを使う。

（鶏と牛は、動物つながりでもあり、食材つながりでもある。）



プログラム

```
interface Animal {
        void cry();
}
```

```
interface Cooking {
        void dish();
}
class Cock implements Animal, Cooking{
        public void cry() {
                System.out.println(“cockadoodledoo”);
        }
        public void dish() {
                System.out.println(“karaage”);
        }
}
class Cow implements Animal, Cooking {
        public void cry() {
                System.out.println(“moooo”);
        }
        public void dish() {
                System.out.println(“sukiyaki”);
        }
}

class TestInterface {

        public static void main(String[] args) {
                Cock yaki= new Cock();
                Cow don= new Cow();
                yaki.cry();
                yaki.dish();
                don.cry();
                don.dish();
        }
}
```

補足）インタフェースと abstract の違い

インタフェースを実装したクラスでメソッドの本体を定義すると言うことを考えると、abstract クラスを継承したクラスで abstract メソッドの本体を定義するというのとよく似ています。インタフェースと abstract クラスには以下に示すような違いがあります。

- インタフェースではどのメソッドも本体を定義することはできませんが、abstract クラスには本体を定義したメソッド（abstract でないメソッド）を含めることができます。
- インタフェースは複数実装できますが、abstract クラスは１つのみしか継承できません（多重継承がサポートされていないため）。
- abstract クラスは継承したクラスとの間にクラス関係を意識する必要がありますが、インタフェースは実装したクラスとの間にクラス関係を意識する必要はありません。

情報科学実験Ｉ　第３回　スレッドとスレッドの同期

１．スレッド（thread）

今まで作ってきたプログラムの処理流れは、一本の糸って感じだった。しかし、応用によっては複数の糸で書いた方が自然なものとなる場合がある。例えば、

例１：画面上に時計を表示しながら、計算処理をする場合、時計スレッドと計算処理スレッドを独立に書く。

例２：１つのスレッドがデータを生成し、１つのスレッドがそれを消費するケース。

Java では main()が１つのスレッドになっている。Java はスレッドを生成して複数のスレッドを平行して実行されているように見せかけることができる。複数のスレッドを並列に実行すると言ってもＣＰＵは１つなので各スレッドを時間を区切ってかわるがわる実行していく。

２．スレッドの生成

スレッド生成法は２つあります。ここでは Java に用意されている Thread クラスを継承して生成する方法を示します。もう一つの方法は付録Ａ参照。

スレッドを新しく生成する場合は、次のようにします。

- スレッドにするクラスを、Java の Thread クラスを継承して作ります。
- スレッドの中身を、run() メソッドに書く。
- スレッドのオブジェクトを生成する。
- start() メソッドを呼び出し、スレッドを起動する。

```
class ThreadTest {
    public static void main(String[] args) {
        Suredo t = new Suredo();
        t.start();
        for (int i = 0; i < 1000; i++) {
            System.out.print('.');
        }
    }
}

class Suredo extends Thread {
    public void run() {
        for (int i = 0; i < 1000; i++) {
            System.out.print('o');
        }
    }
}
```

３ スレッドの状態

java のスレッドは、生成されてから消滅するまでにいくつかの状態を示していきます。

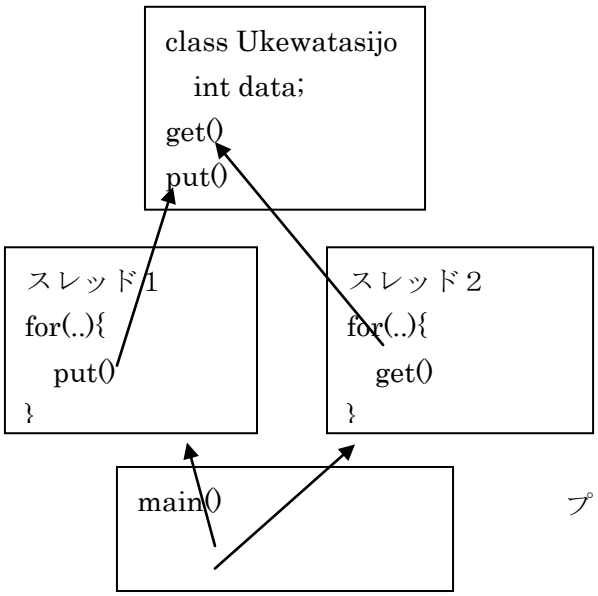
(1) 初期状態：　新しいスレッドオブジェクトを生成した状態です。この状態ではまだスレッドは動作していません。start() メソッドにより、実行可能状態となります。

(2) 実行可能状態:　スレッドの start()メソッドを呼び出されて run()メソッドを呼び出しスレッドが動作している状態です。この状態は、さらに実行中と実行時間割り当て待機状態に分けられます。実行中は、時間分割処理によって CPU の実行時間が割り当てられ run()メソッドの処理が実行されている状態です。CPU が１つしかないコンピュータでは、実行中のスレッドは１つだけしか存在しません。その他のスレッドは実行時間割り当て待機状態で、CPU 時間が割り当てられるのを待機しています。
(3) ブロック状態：　システム上の他の操作や、スレッドの排他制御や同期処理などにより、スレッドの動作が割り込み処理により一時的にブロックされている状態です。ブロック状態となっているための他の処理が修了すれば、また実行可能状態に戻ります。スレッドは動作中に何度も実行可能状態とブロック状態を行き来することができます。
(4) 終了状態
スレッドの処理が終了した状態です。一度終了状態になったスレッドは、再び動作を開始することはありません。

課題３－１：前回、FIFO と LIFO を作りました。前回はデータをサイズ３の配列に入れていきましたが、配列を１つの変数に置き換えてしまう（サイズ１の配列と思ってもよい）と FIFO も LIFO も同じ構造になります。これを Ukewatasijo というクラスにして get()と put()というメソッドを用意します。

また、２本のスレッドを作り、一方のスレッドは 1ms に一度バッファにデータを１つづつ（1,2,3,…1000）を put()し、もう一方にスレッドは 1ms に一度バッファからデータを取り出(get())して表示します。このプログラムを作成しなさい。

このプログラムは２つのスレッドがデータをうまく受け渡せないのを確認するするものです。うまく受け渡すにはスレッド間の同期という仕掛けが必要で、これは第６回で学習します。



プログラムの構造

このプログラムは色々考えられます。一番簡単なのはmainのクラスでUketorijoをstaticに宣言してしまう方法です。mainのクラスは次のようになります。

```
class Ukewatasijo{
       :
 }
class Okurite extends Thread{
    public void run(){
         :
    }
}
class Ukete extends Thread{
    public void run(){
         :
    }
}
class Thread1{
    public static Ukewatasijo hako = new Ukewatasijo();
    public static void main(String arg[]){
        Ukete uket = new Ukete();
        uket.start();
        Okurite okuri = new Okurite();
        okuri.start();
    }
}
```

(予備知識)上の方式の場合、Thread1でstaticに宣言したオブジェクトbuffは、外部のクラス（Sender,Receiver）では、Thread1.buffとしてアクセスできる。

本格的プログラムにするには、
Uketorijoのオブジェクトをmain()で作り、OkuriteとUketeに送ってやると、もっとよいかもしれません（以下参照）。

```
class Uketorijo{
        …
        get(){…}
        put(…){…}
}
class Okurite …{
        Uketorijo hakoo;
        Okurite(…){…}
        public void run(){…}
}
class Ukete …{
        Uketorijo hakou;
        Ukete(…){…}
        public void run(){…}
}
public class Thread2{
              public static void main(String[] args){
                            Ukewataisjo hako = new Ukewatasijo();
                            Ukete uket= new Ukete(hako);
                            uket.start();
                            Okurite okuri= new Okurite(hako);
                            okuri.start();
```

```
                }
}
```

４．スレッドの同期

Act1：複数のスレッドが協力する場合、時々連絡（スレッド間の同期）を取り合う必要があります。様々な同期の基本型を全て含んでいるのが生産者消費者問題です。

◎生産者消費者問題（Producer-Consumer Problem）：

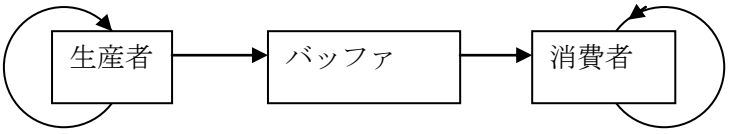
生産者はデータを作ってはバッファへ送る、という作業を繰り返す。

消費者はバッファからデータを取り出しては使ってしまう。

生産者はデータをバッファに送ろうとした時バッファが満杯だと自主的に待ちに入る。消費者はバッファからデータを取ろうとした時バッファーが空なら自主的に待ちに入る。

生産者はデータを送ったとき待っている消費者がいると、データの準備が出来た旨を連絡し待ちから出す。

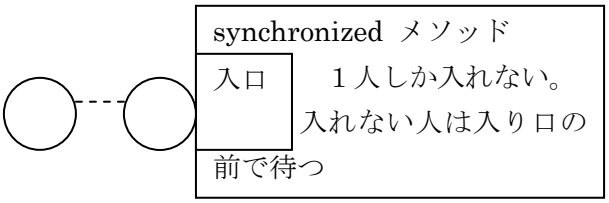
消費者はデータを取り出したとき待ている生産者がいると、バッファが空いた旨連絡し待ちから出す。



Act２：Java の持っている同期のための機構（イメージトレーニング）

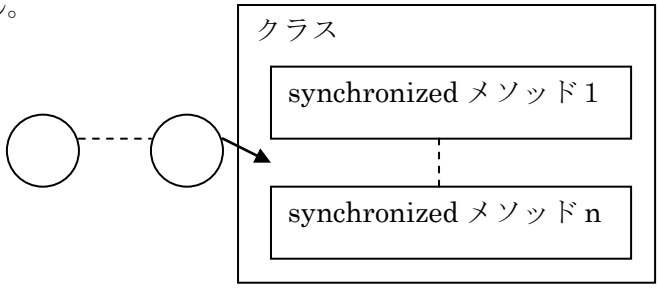
A：１時に１人しか使えないようにする機構（１つしかないものの奪い合いで入り口に待ち行列ができます。

(A1) synchronized メソッド、sunchronized ブロック：

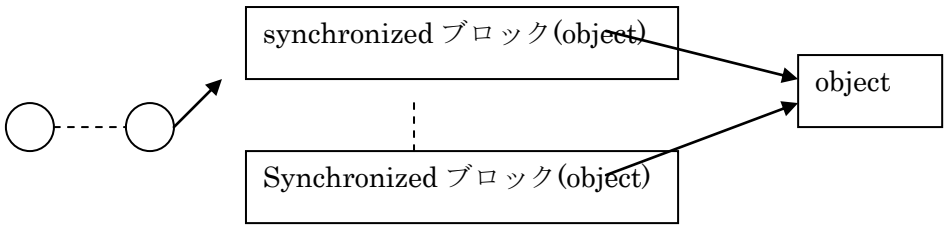


(A2)synchronided メソッド、synchronized ブロックの択一動作機構：

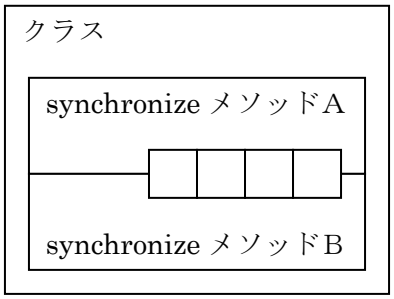
◎１つのクラス内に複数の synchronized メソッドがある場合、１時には１つのメソッドにしか入れません。例えば下図でメソッド１に誰かが入ると、他のスレッドには誰もはいれません。



◎ オブジェクトつながりの synchronized ブロックも、一時には１つのブロックにしか入ることはできません。



B：共同作業のための機構（声をかけあう関係）：意図的にプログラムで待ち行列を操作します。



クラス内に複数の synchronized メソッドがある場合、それらで共有する待ち行列が用意されます（上図内の正方形）。これを使って複数のスレッドが共同作業（同期）を行います。synchronized メソッドにはいれた人は、メソッド内で作業を行い、その結果①メソッドを出るか、②wait()命令を出して自ら上図中の待ち行列に入るか、ということになります。これにより、メソッドのロックが解けて他のスレッドが synchronized メソッドを使えるようになります。この待ち行列には、どちらのメソッド（上図A，B）からも入ることができます。待ちに入ったままでは困るので、待ちから出す命令も用意されています。notify()命令は待ちの先頭のスレッドを実行可能状態にします。また notifyAll()命令は、待ちに入っているスレッド全部を実行可能状態にします（実際には待ちの先頭から順に実行していきます）。
同様の待ち行列は synchronized ブロックにも用意され、同様の命令が使えます。

Act３：上の道具を使って生産者消費者モデルをどうプログラムするのか？
先ず、生産者クラス、消費者クラス、バッファクラスを用意します。バッファクラスには生産者がデータをバッファに置くために使うメソッド synchronized　put(object)と、消費者がバッファからデータを取り出すために使うメソッド synchronized get()を用意します。これにより put()と get()は同時には動かなくなります（同時に動くと、データ書き込み中に取り出しをしたりしてめちゃめちゃになる）。次はメソッド内での行動です。生産者はバッファが空いていなければ wait()を出して待ちに入ります。バッファが空いていればバッ

ファにデータ入れ、さらにデータ到着待ちの消費者に notify()を出して待ちから出してやります。消費者はこの逆で、バッファが空なら wait()で待ちに入ります。また、バッファにデータがあれば取り出し、バッファが満杯で待っている生産者に notify()を出して待ちから出してやります。

Act４：プログラム例（同期していないプログラム）

```
class Buffer{
        int data;
        int get(){
                return data;
        }
        void put(int a){
                data= a;
        }
 }
class Producer extends Thread{
    public void run(){
                for( int i = 0 ; i < 100; i++ ){
           try{
                        Thread.sleep(1);
           }catch(Throwable t){}
            PCproblem_as.buff.put(i);
                }
    }
}
class Consumer extends Thread{
    public void run(){
        for( int i = 0 ; i < 100 ; i++ ){
             try{
                                Thread.sleep(1);
             }catch(Throwable t){}
                System.out.println(PCproblem_as.buff.get());
        }
    }
}
class PCproblem_as{
    public static Buffer buff = new Buffer();
    public static void main(String arg[]){
        Consumer cons = new Consumer();
        cons.start();
        Producer prod = new Producer();
        prod.start();
    }
}
```

一方のスレッドが i を書き出す前にもう一方が i を取り出すとおかしくなる。
synchronized メソッドを使えば、この問題は解決する。

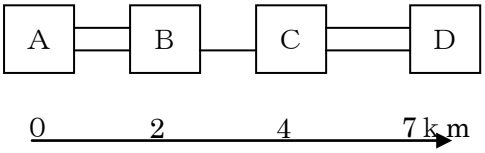
プログラム例　生産者消費者プログラム（同期あり）

```
class Buffer    {         // バッファ
   private int buffer=9999;
   private boolean full=false;
```

```
   public synchronized void put(int x)  {//バッファが満なら待つ
   try{
     while (full==true) wait();  // 何故"if"でなく"while" か?
     buffer= x;
     full= true;
     notifyAll();  // どんな時"notify()"でなく"notifyAll()"か?
     } catch (InterruptedException ie) {}
   }

   public synchronized int get()  { //バッファが空なら待つ
     int result = 0;
     try{
       while (full==false) wait();
       result = buffer;
       full=false;
       notifyAll();
     } catch (InterruptedException ie) {}
     return result;
   }
}

class Producer extends Thread { // 生産者
    private int x = 0; // value to be "produced"
    public void run(){
        for(int i=0; i<100; i++)
                PCproblem.buf.put(x++);
    }
}

class Consumer extends Thread{  //消費者
    private int x = 0;
    public void run(){
                for(int i=0; i<100; i++){
                        x= PCproblem.buf.get();
                        System.out.println("consume"+x);
                }
    }
}

public class PCproblem {
   public static Buffer buf=new Buffer();
    public static void main(String[] args)    {
        Producer t1 = new Producer();
        Consumer t2 = new Consumer();
                t1.start();
                t2.start();
    }
}
```

注）生産者と消費者が複数いる時、上記のように notifyAll()を使う必要がある。理由：待ち行列は生産者消費者共通なので、タイミングによって消費者の notify()が消費者のスレッドを起動するケースが起こる。例えば get,get,の後 put が２つ同時に起こった場合、後の put が待ちに入り、さらに get が来たりすると起こすべきスレッドの順序が狂いおかしくなる。

課題３－２：右図は鉄道の路線図である。A,B,C,D４駅があり、BC 間は単線、その他は複線になっている。車両は２台で、それぞれ AD 間を往復運転している。BC 間での衝突を防ぐ道具として、路線に１つだけ Staff（通行手形）というものがあり、普段は B 駅かC駅に置いてある。BC 間を走行したい車両は駅で Staff を受け取り単線に入り、単線を出る時駅に置いていく。駅に Staff がない場合、もう１台の車両の到着を待つ。２台の車両は最初 A 駅と D 駅からスタートする。車両の速度は時速 1km とし、50 時間の間に、各車両が待ちに入った時刻と、待ちから出た時刻を表示せよ。



付録Ａ： スレッドを Runnable インタフェースで生成する方法

Java は一つのスーパークラスしか継承できず多重継承することはできません。したがって、他のあるクラスを継承しているサブクラスをスレッドとして実行したい場合は、 Thread クラスを継承することは多重継承になるのできません。

このような場合は、 Runnable インタフェースを実装します。実は Thread クラスも Runnable インタフェースを実装しています。したがって、すべてのスレッドクラスは Runnable インタフェースの実装クラスです。

スレッド

```
class ImRun implements Runnable{
  String str;
  ImRun(String s){
    this.str=s;
  }
  public void run(){
    for(int i=1; i<=5; i++){
    System.out.println(str+":"+i);
    }
  }
}
class ThTest{
  public static void main(String args[]){
    ImRun imrun1=new ImRun("A");
    ImRun imrun2=new ImRun("B");
    Thread thre1=new Thread(imrun1);
    Thread thre2=new Thread(imrun2);
    thre1.start();
    thre2.start();
  }
}
```

付録Ｂ：　スレッドを　Runnable で生成した時の生産者消費者プログラム

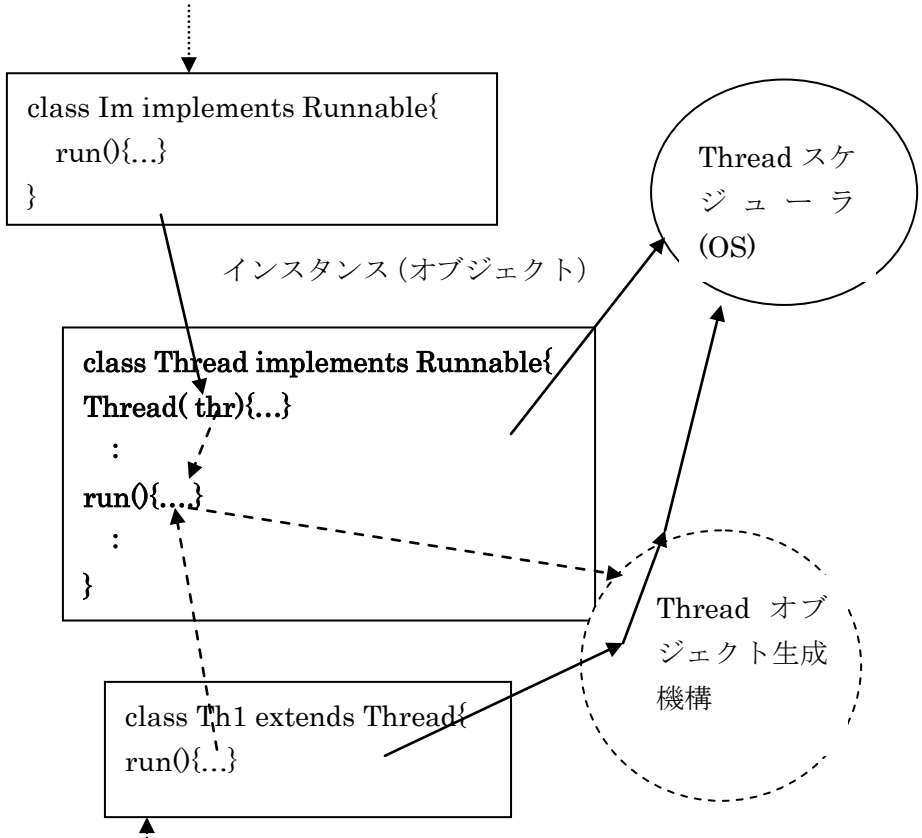
```
class Buffer        {     // バッファ
    private int buffer=9999;
    private boolean full=false;

    public synchronized void put(int x)  {//バッファが満なら待つ
    try{
       while (full==true) wait();   // 何故"if"でなく"while"  か?
       buffer= x;
       full= true;
       notifyAll();   // どんな時"notify()"でなく"notifyAll()"か?
       } catch (InterruptedException ie) {}
    }

    public synchronized int get()  {//バッファが空なら待つ
     int result = 0;
       try{
          while (full==false) wait();
          result = buffer;
          full=false;
          notifyAll();
       } catch (InterruptedException ie) {}
       return result;
    }
}

class producer implements Runnable{        // 生産者
      private int x = 0; // value to be "produced"
      public void run(){
            for(int i=0; i<100; i++) PCprob.buf.put(x++);
      }
}

class consumer implements Runnable {      //消費者
      private int x = 0;
      public void run(){
            for(int i=0; i<100; i++){
                        x= PCprob.buf.get();
                        System.out.println("consume"+x);
            }
      }
}

public class PCprob {
      public static Buffer buf=new Buffer();
      public static void main(String[] args){
            Thread t1 = new Thread(new producer());
            Thread t2 = new Thread(new consumer());
            t1.start(); t2.start();
      }
}
```

補助説明：　スレッド裏機構古谷イメージ

（１）Runnable の場合：

```
Im im1=new Im ("A");
Thread thr=new Thread(im1);
thr.start();
```

Im のオブジェクト（インスタンス）を、（Java に用意されている）Thread クラスのコンストラクタに引数として渡すと、ユーザが書いた run 本体が Thread の run に渡され、その Thread のインスタンスがスレッドスケジューラ（古谷が仮に付けた名前）に送られる。プログラムは複雑になるが、スレッドオブジェクトを作る様子が分かる。



（２）Thread の継承

```
ThT th1 = new ThT();
th1.start();
```

Th1 の run が(Java の)Thread クラスの run に上書きされた形で Thread オブジェクト生成機構（古谷が仮に名づけた）に渡され、オブジェクト生成機構でスレッドオブジェクトに作り上げられ Thread スケジューラに送られる。プログラムは簡単になるが、裏でスレッドオブジェクトを作る機構が働いている。

第４回　GUI（グラフィックユーザインタフェース）

次のプログラムは、ウインドウを開き、そこに３つの押しボタンが付いている。各ボタンは、(1)文字列の表示、(2)正方形の表示、(3)円の表示を行う。(3)はボタンを押すごとに円が右に移動する。

（プログラム１）

```
import java.awt.*;
import java.awt.event.*;
public class F6 extends Frame implements ActionListener{
	int pc= -1;
	//主処理
	public static void main(String args[]){
		Frame f=new F6();
		f.setTitle("ボタンクリックで文字列表示");
		f.setSize(640,400);
		f.setVisible(true);
	}
	//部品セット
	F6(){
		setLayout(new FlowLayout());
		Button b0=new Button("000");
		Button b1=new Button("111");
		Button b2=new Button("222");
		b0.addActionListener(this);
		b1.addActionListener(this);
		b2.addActionListener(this);
		add(b0);
		add(b1);
		add(b2);
		addWindowListener(new WinAdapter());
	}
	//閉じる
	class WinAdapter extends WindowAdapter{
		public void windowClosing(WindowEvent we){System.exit(0);}
	}
	//描画
	public void paint(Graphics g){
		String s="hello this is a string";

		if(pc==0){
			g.drawString(s,100,150);
		}
		if(pc==1){
			g.drawRect(50,50,30,30);
		}
		if(pc==2){
			g.drawOval(100,200,10,10);
		}
	}
	//イベント
	public void actionPerformed(ActionEvent ae){
		if(ae.getActionCommand()=="000"){
			pc=0;
			repaint();
		}
		if(ae.getActionCommand()=="111"){
```

```
                pc=1;
                repaint();
            }
            if(ae.getActionCommand()=="222"){
                pc=2;
                repaint();
            }
        }

//repaint()で画面クリヤーしないようにするには以下を 3 行を生かす
//      public void update(Graphics g){
//          paint(g);
//      }

}
```

上記プログラムの解読ヒント集

１、Frame にボタンを付け、ボタン push イベントを捕まえるには main クラスのインスタンスを作ります。f= new F6();

２、implements ActionListener はイベントが発生した時の処理を、actionPerformed に書け、という指定だと思え。

３、ボタンを作るには、Button btn = new Button("Button");とします。

これだけではボタンは作成できますが、ボタンを押した時に Event を発生させることができません。Event を発生させるには、addActionListener メソッドを使って、このボタンが ActionEvent を発生させますよと書いておく必要があります。よって書き方としては、

btn.addActionListener(this);

のような形になります。addActionListener の()の中には、インスタンスを指定する必要がありますので"this"を書いておいて下さい。

これでボタンが押された時に ActionEvent が発生し、この ActionEvent を受け取ることがこのインスタンスではできるようになります。最後に Event を受け取った時に何をするのかを書いておきます。

ActionEvent を受け取った時には、

```
public void actionPerformed(ActionEvent e){
  ....
}
```

というメソッドが呼びだされます。この中に ActionEvent が発生したときに行いたいことを具体的に書くようにします。

４、ウインドウ右上の x ボタンを押すと、ウインドウが閉じるようにするために、WindowAdapter を用意する。

５、paint()は、フレームが表示される時と、repaint()命令で実行される。

６、Graphics g とは、お絵かきセットつき画面 g と考えてください。

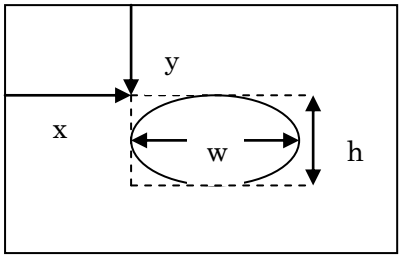
７、repaint()を実行すると、システム内で update()が呼ばれる。update は先ず画面をクリヤーした後 paint()を呼んで図形を描く。故に、paint()しかない update を作っておけば、

描画履歴が残る。
8、Math.random()は 0～1 の間の乱数を返す

課題０：maru と sikaku というボタンを作り、それぞれの形を表示するようにしよう。
課題１：大きさの同じ円を５つ並べて表示するボタンを作れ。
課題２：３つの同心円を描くボタンを作れ。
課題３：８×８の碁盤の目を描け。
課題４：ラジコンカーを操作するシミュレーション。ボタンを３つ（left, up, right）用意する。最初画面下の方に四角（車）を置いておき、up ボタンで上に、left ボタンで左に、right ボタンで右に動くようにしよう。
課題５：ボタンを押すごとに、(4 近傍へ）ランダムウオークする円を描け。乱数で動く方向を決める（過去の円は消さない）今日の課題 “update”を意識して。

発展課題１：円や線を繰り返し使い、色も色々変えて美しい画面を作れ。

描画命令
１．円や楕円を描くには：g.drawOval(x,y,w,h)
２．直線は：g.drawLine(x1,y1,x2,y2)
ここで x1,y1 は始点の座標、x2,y2 は終点の座標
３．四角は：g.drawRect(x,y,w,h);



＊色はペンの色と考える、１度持つ(g.setColor)
と、次の g.setColor までペンの色は変わらない。
４．色指定：g.setColor(色名)
ここで色名は、Color.red,Color.magenta,
Color.blue,Color.cyan,Color.pink なんて指定
５．色指定：g.setColor(new Color(赤成分,緑成分,青成分));
ここで、赤、緑、青成分は０～２５５の整数
５．塗りつぶしは g.fillOval(…), g.fillRect(…)等

＜解説＞　　テキストボックス
文字を入力したり、出力したりする箱をテキストボックスという。次のプログラムはフレームにテキストボックス２つとボタンを用意し、左のボックスに数値を書き込んでボタンを押すと、右のボックスに左のボックスの２倍数値を表示する。

発展課題２：３つのテキストボックス（光の３原色に対応し r,g,b）を用意する。3 つのボックスにそれぞれ赤、緑、青の成分（０～２５５）を書き込んでボタンを押すと、それに対応する色の塗りつぶし円が表示されるようにせよ。

（プログラム２）

２倍の数値を表示するプログラム

```
import java.awt.*;
import java.awt.event.*;
public class F61 extends Frame implements ActionListener{
        TextField inp, outp; //入出力用テキストフィールド
    //主処理
    public static void main(String args[]){
        Frame f=new F61();
        f.setTitle("ボタンクリックで整数を２倍にします");
        f.setSize(640,400);
        f.setVisible(true);
    }

    //部品セット
    F61(){
        setLayout(new FlowLayout());
        inp= new TextField("0",5);
        outp= new TextField("999",5);
        add(inp); //テキストフィールドの貼り付け
        add(outp); //テキストフィールドの貼り付け
        Button b0=new Button("000");
        b0.addActionListener(this);
        add(b0); //ボタンの貼り付け
        addWindowListener(new WinAdapter());
    }

    //ウインドウを閉じる時の処理
    class WinAdapter extends WindowAdapter{
        public void windowClosing(WindowEvent we){System.exit(0);}
    }

    //ボタンを押された時のイベント処理
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand()=="000"){
                String in_text= inp.getText();
                 // getText は Text フィールドの文字列を取り込む命令
                int in_data=Integer.parseInt(in_text);
                 // Integer.parseInt は文字列を整数に直す命令
                in_data= in_data *2;
                outp.setText(String.valueOf(in_data));
                 // setText は Text フィールドに文字列を書き込む命令
                 // String.valueOf は整数を文字列に変換する命令
        }
    }
}
```

情報科学実験Ⅰ　第５、６回　アプレット
第５回　アプレット入門
今までの Java プログラムは main()というメソッドがありました。このようなプログラムはアプリケーションと呼ばれます。これに対し、ホームページに張り込んだりできる比較的小さなプログラムをアプレットと言います。アプレットではウインドウの生成作業が不要で、GIU が比較的簡単になっています。ただ、ファイルへの入出力が出来ないなど、多少の制限はあります。

Applet の処理の流れは、ビューアにより,init(),start(),paint(),update(),stop(),destroy()等の関数がこの順に呼ばれるので，プログラマはこれらの関数の内部の実行部を記述すれば良い。これが分かっているとプログラムが簡単になる。

init ()：アプレット起動時に呼ばれる。初期化プログラムを記述する。

start()：アプレットがアクティブになった時呼ばれる。起動プログラムを記述する。

paint()：初期グラフィック表示プログラムを記述する。

update()：グラフィック表示の変更等のプログラムを記述する。
利用者プログラム内で記述した repaint()で java システム内の update()が呼ばれる。java システム内の update()は画面をクリヤ（消）して、paint()を呼ぶ。画面を消したくなければ自分で paint()しか呼ばない update()を書く（システムの update()に上書きされる)。

stop()：アプレットが非アクティブになった時呼ばれる。アプレットの停止プログラムを記述する。

destroy() : アプレットが閉じられた時呼ばれる。アプレットの開放プログラムを記述する。

これらの関数は必要に応じ一部省略することもできる。

アプレットプログラムのコンパイルと実行
一番上のコメント文/*.....*/は意味があるので消さないこと。これがあることにより、applet のコンパイルと実行は次のようになる。
>javac　~~.java　　//コンパイル（アプリケーションと同じ）
>appletviewer　~~.java　　//実行

描画命令

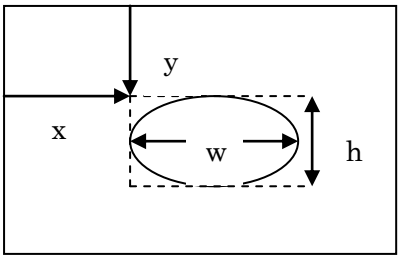
６．円や楕円を描くには：drawOval(x,y,w,h)

７．直線は：drawLine(x1,y1,x2,y2)

ここで x1,y1 は始点の座標、x2,y2 は終点の座標

８．色指定：setColor(色名)

ここで色名は、Color.red,Color.magenta,
Color.blue,Color.cyan,Color.pink なんて指定



プログラム例５－１　円を 4 つ並べて描く

```
/*<applet code="Circles.class" width=300 height=300></applet>*/
import java.applet .*;
import java.awt.Graphics;
public class Circles extends Applet {
        public void paint(Graphics g){
            int i;
            for(i=20; i<100; i=i+20)
                g.drawOval(i,50,20,20);
         }
}
```

プログラム例５－２マウスのボタンを押すと円を描くプログラム

マウスボタンに反応して動くアプレットを作るには頭に implementsMouseListener と書いて、init()に addMouseListener(this)を書き、マウスボタン操作への反応をメソッドとして書く。

```
/*<applet code="Mcircle.class" width=300 height=300></applet>*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Mcircle extends Applet implements MouseListener{
        int x=-10, y=-10;
        public void init(){
            addMouseListener(this);
        }
        public void mousePressed(MouseEvent e){
```

```
            x=e.getX();
            y=e.getY();
            repaint();
        }
        public void mouseClicked(MouseEvent e){}//これらは使わなくても書く
        public void mouseEntered(MouseEvent e){}//
        public void mouseExited(MouseEvent e){}//
        public void mouseReleased(MouseEvent e){}//
        public void paint(Graphics g){
                g.drawOval(x-5,y-5,10,10);
         }
}
```

注）repaint()は paint を呼ぶが、その前に画面を消す。

課題

１．画面上で５回クリックする。5 回目に 5 つのクリック点を（クリックした順に）結ぶ折れ線を表示する。

２．海面近くを漂う（上から見えない）機雷を上空から爆撃し除去する。機雷周辺距離２０ｍに爆弾を落とすと機雷が爆発して取り除ける（機雷周辺２０ｍが赤くなる)。機雷から２０～５０ｍの範囲に爆弾が落ちると落ちた地点を中心に黄色い円ができる。機雷は流され少しずつランダムに動く。

（機雷と着弾点の間の距離を求めるのに 3 平方の定理と、Math.sqrt(距離の自乗)を使用）

３．複数（３つ）の機雷があるケースをシミュレーションするため、機雷クラスを使う方法を検討する。次に機雷クラスの配列を使ったプログラムを示す。ただし、プログラムでは１つの機雷しか生成していない。これを変更し３つの機雷をシミュレートできるようにせよ。

＜課題３のヒントプログラム＞

```
/*<applet code="Sokai.class" width=300 height=300></applet>*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Sokai extends Applet implements MouseListener{
        int   x=100, y=100, atari=0;
        double dist;
        Kirai kirais[]= new Kirai[3];

        public void init(){
                kirais[0]= new Kirai();
            addMouseListener(this);
        }
```

```
        public void mousePressed(MouseEvent e){
            x=e.getX();
            y=e.getY();
            atari=0;

dist=Math.sqrt((x-kirais[0].xo)*(x-kirais[0].xo)+(y-kirais[0].yo)*(y-kirais[0].yo));
            if((dist<50)&&(dist>20))
                        atari=1;
            if(dist<=20)
                        atari=2;
                repaint();
                kirais[0].move();
        }
        public void mouseClicked(MouseEvent e){}//これらは使わなくても書く
        public void mouseEntered(MouseEvent e){}//
        public void mouseExited(MouseEvent e){}//
        public void mouseReleased(MouseEvent e){}//
        public void paint(Graphics g){
                if(atari==0) g.setColor(Color.pink);
                if(atari==1) {
                        g.setColor(Color.yellow);
                        g.drawOval(x,y,20,20);
                }
                if(atari==2) {
                        g.setColor(Color.red);
                        g.drawOval(kirais[0].xo,kirais[0].yo,20,20);
                }
         }
}

class Kirai{
        int xo, yo;
        Kirai(){
                xo=(int)(Math.random()*300);
                yo=(int)(Math.random()*300);
        }
        void move(){
                int d= (int)(Math.random()*4);
                switch (d){
                        case 0: xo= xo+5; break;
                        case 1: xo= xo-5; break;
                        case 2: yo= yo+5; break;
                        case 3: yo= yo-5; break;
                        default: System.out.println("err");
                }
        }
}
```

第６回　アプレットでアニメーション

Applet でアニメーション（下のプログラム読解のヒント）

１、implements Runnable は run()というメソッドを自作しますよって意味

２、アニメーションは普通 implements Runnable を使ってスレッドを作って実行する（sleep の間入力が効かなくなったり、エラー時にロックするのを防ぐため)。

３、(implements Runnable 付きの）アプレットは、init()、start()、paint()、run()の順に実行される。paint()は、最初のアプレット表示時(上の行)と、repaint()で実行される。

４、アニメは｛表示、お休み(sleep)、表示、お休み(sleep)、、、、、｝を繰り返す。

円が右へ移動するプログラム

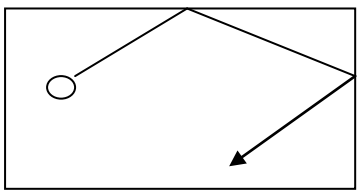
```
/*<applet code="Ball.class" width=300 height=300></applet>*/
import java.applet.*;
import java.awt.*;
public class Ball extends Applet implements Runnable{
    int x;
    Thread th;
    public void start(){
        th= new Thread(this);
        th.start();
    }
    public void run() {
        for(x=0; x<180; x++) {
            repaint();
            try{
                Thread.sleep(100);
             }catch(InterruptedException e) {   }
        }
    }

    public void paint(Graphics g) {
          g.drawOval(x,90,20,20);
          g.drawLine(0,110,200,110);
    }
}
```

課題１、上のプログラムを参考に、ボールを左右に大きく往復運動するようにしよう。テーブル右端にきたら、落下するようにしよう。

課題２、上の往復運動を無限に繰り返す（for(;;)を使用）ようにしよう。

課題３、上の水平移動を、机の上をころがるボールと考え、右端に来たら落下させてみよう。(できる人は落下の際、加速度を付けてみよう)

課題４、　ビリヤード：長方形の枠内にボール（円）があります。ボールが枠で跳ね返りながら移動する様子をシミュレートしよう。



ヒント：ボールの進む方向を示す変数、
例えば dx=1 なら右方向、dx=-1 なら左方向、
とするとボールの跳ね返りが簡単に表せるかも。

発展課題：以下は、 ３台の車がレースをするプログラムである。これを参考に、課題 4 のボールクラスを作り３つのボールがビリヤード風に動くようにせよ（ボール同士の衝突は無視）。

```
/*<applet code="Race22.class" width=300 height=300></applet>*/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Race22 extends Applet implements Runnable {

    Car cars[]= new Car[3];
    Thread th;

    public void start() {
        for(int i=0; i<3; i++)
                    cars[i]= new Car(i);
        th= new Thread(this);
        th.start();
    }

    public void run() {
        try {
              for(int i=0; i<50; i++) {
                  for(int j=0; j<3; j++)
                              cars[j].move();
                  repaint();
                  th.sleep(100);
              }
         }
         catch(InterruptedException e){}
     }

    public void paint(Graphics g) {
          for(int j=0; j<3; j++) {
                    g.drawOval(cars[j].x, cars[j].y, 20, 20);
          }
    }
}

class Car{
          int x,y, number;

          Car(int number) {
                    x= 0;
                    y= number*50+50;
                    number++;
        }

          void move() {
                    x= x+ (int)(10 * Math.random());
        }
}
```