

# Javaプログラミング(10) スレッド(2)

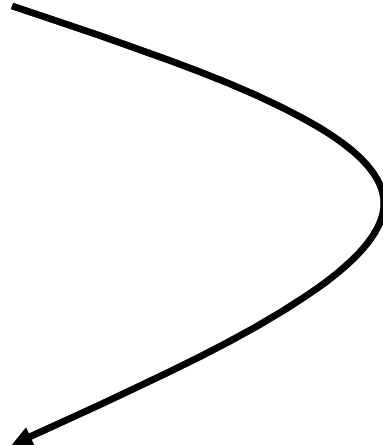
成蹊大学理工学部  
情報科学科

# 期末試験について

- 1月27日(金) 2限
- 持込物:なし
- テスト範囲
  - 14回までの内容

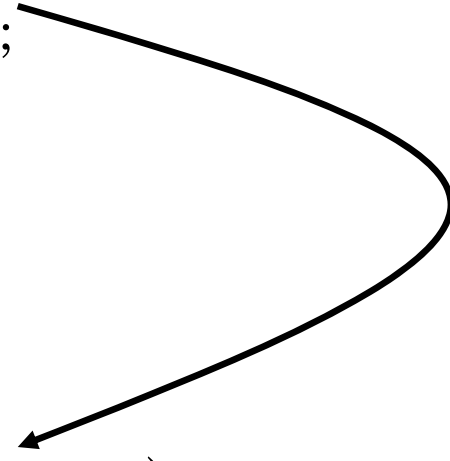
# Threadクラスを用いた例（復習）

```
public class CountTestA {  
    public static void main(String[] args) {  
        CountA threadA = new CountA(); // (3) Threadのインスタンス生成  
  
        for (int i=0; i<10; i++) {  
            System.out.println("main: "+i);  
        }  
    }  
}  
  
class CountA extends Thread {  
    {  
        for (int i=0; i <= 10; i++) {  
            System.out.println("Thread: " + i*10);  
        }  
    }  
}
```



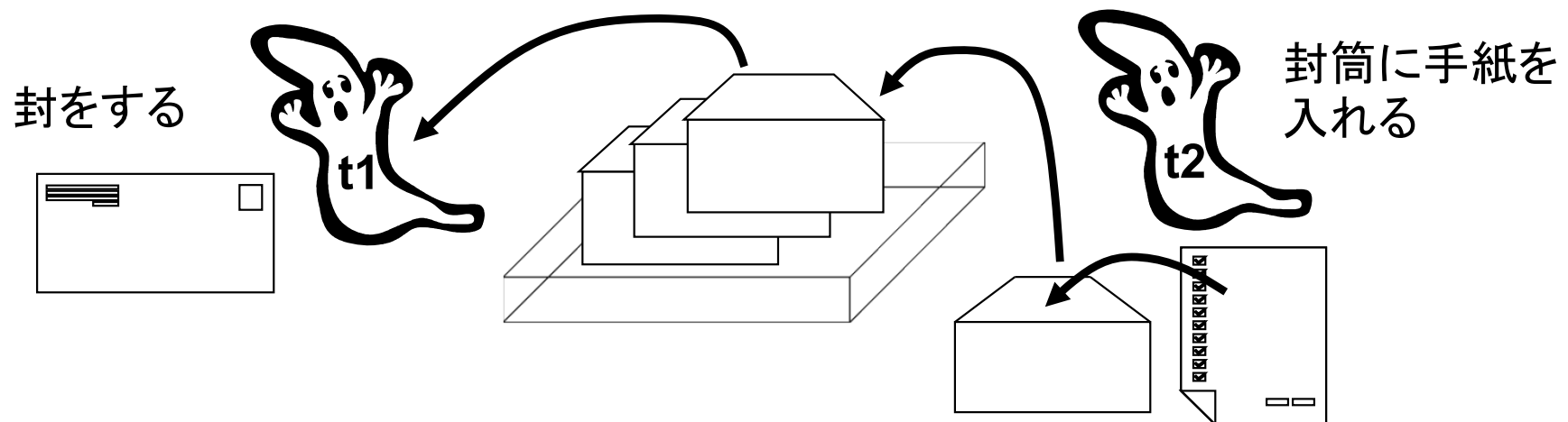
# Runnableインタフェース実装例 (復習)

```
public class CountTestB {  
    public static void main(String[] args) {  
        CountB ctB = new CountB(); // (3) インスタンスを作成  
  
        for (int i=0; i<10; i++) {  
            System.out.println("main: "+i);  
        }  
    }  
}  
  
class CountB implements Runnable {  
    {  
        for (int i=0; i <= 10; i++) {  
            System.out.println("Thread: " + i*10);  
        }  
    }  
}
```

A curved arrow originates from the 'main' loop's 'println' statement and points to the 'Thread' loop's 'println' statement, indicating a flow or comparison between the two loops.

# スレッド同士の待ち合わせ

- 2つのスレッドが協調して仕事をするためには、お互いに相手の作業状況に合わせて自分の処理を行う必要がある。
- (例) キューを共有した2スレッド間の協調  
生産者－消費者問題
  - － 生産者: 封筒に手紙を入れる係
  - － 消費者: 封をする係
  - － キュー: 作業対象となる手紙を入れる箱  
箱が空だと封をする係は仕事ができない。箱がいっぱいだと言紙を入れる係はこれ以上手紙を置けない。



# スレッド間の同期をとる方法

## スレッド間の同期を取る方法

```
synchronized aMethod (...) throws InterruptedException {  
    while (!求めている条件) {  
        wait(); //スレッドを待たせる  
    }  
    notifyAll(); //スレッドに通知する (眠っていたスレッドを起こす)  
}
```

# wait, notify, notifyAll

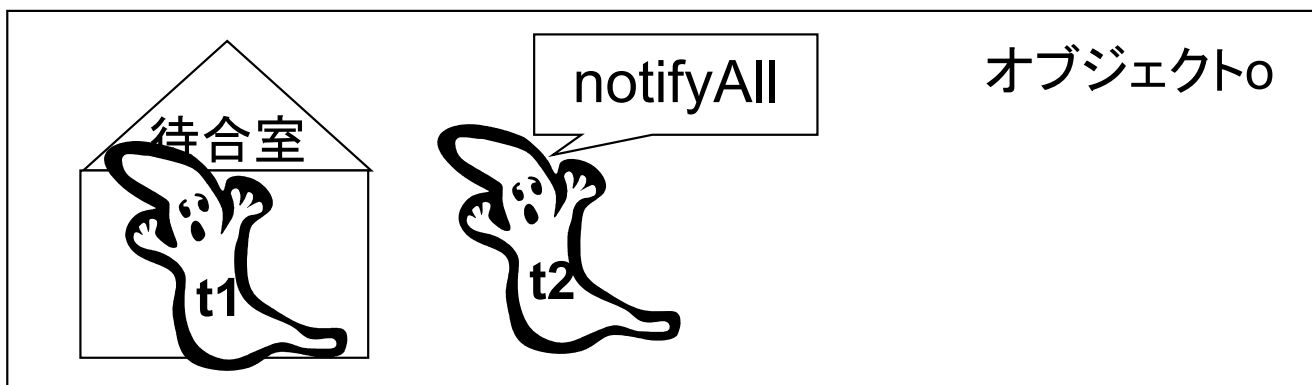
- wait, notify/notifyAllというメソッドはjava.lang.Objectクラスで定義されている。
  - つまり, すべてのオブジェクトがこれらのメソッドを持っている.
- 各オブジェクトには, メソッドwait/notify/notifyAllのために, スレッドの「待合室(休憩室)」が用意されている. この待合室のことを「待機集合(wait set)」と呼ぶ.
- notify/notifyAllはwait()で眠っていたスレッドを起こす
  - notifyメソッドは待機集合の中の1つのスレッドに通知をする
  - notifyAllメソッドは待機集合の中のすべてのスレッドに通知をする.
    - notifyAllの方が確実(「とりあえず全員起こしてみる」という考え).

# スレッド同期(簡単な説明)

- あるスレッドt1があるオブジェクトoのwaitメソッドを呼び出すと、スレッドt1はオブジェクトoの待機集合に入る(待合室で休む)。従って、waitメソッドの中でt1の処理は中断される。



- ここで、別のスレッドt2がオブジェクトoのnotifyAllメソッドを呼び出すと、オブジェクトoの待機集合中のスレッドにそのことが通知される。その結果、待機中であったスレッドが実行可能になる(スレッドが起こされる)。





# スレッドの同期(詳しい説明)

- 各オブジェクトには、同期のためのロック(鍵)と待機集合が1つずつ用意されている。
- あるオブジェクトoのwait/notify/notifyAllメソッドを呼び出す時、呼び出し元は、オブジェクトoのロックを獲得していなければならない。
  - これらのメソッドを呼び出す際には待機集合の状態が変化するため、ロックしておかないと問題が起きる可能性がある。
- 従って、メソッドwait/notify/notifyAllは、同期メソッド(synchronizedメソッド)、あるいは同期ブロック(synchronizedブロック)の中で使わなければならない。
  - 同期メソッドを実行しているスレッドは、オブジェクトoのロックを持っているので、オブジェクトo(つまりthis)のwait/notify/notifyAllを呼び出すことができる。
  - this以外のオブジェクトの待機集合を使う場合には、同期ブロックの中で使う。

# スレッドの同期(Cont.)

- スレッドt1がオブジェクトoのwaitメソッドを呼び出した時, t1はoのロックを持っている. このとき, スレッドt1はoのロックを解放して待機集合に入る
  - もし, スレッドt1がオブジェクトoのロックを持ったまま待機集合に入ってしまうと, 別のスレッドはoのロックを獲得できず, だれもnotifyやnotifyAllを呼び出せなくなる. そうなると, スレッドt1は待機集合中で永遠に待ち続けることになってしまう.



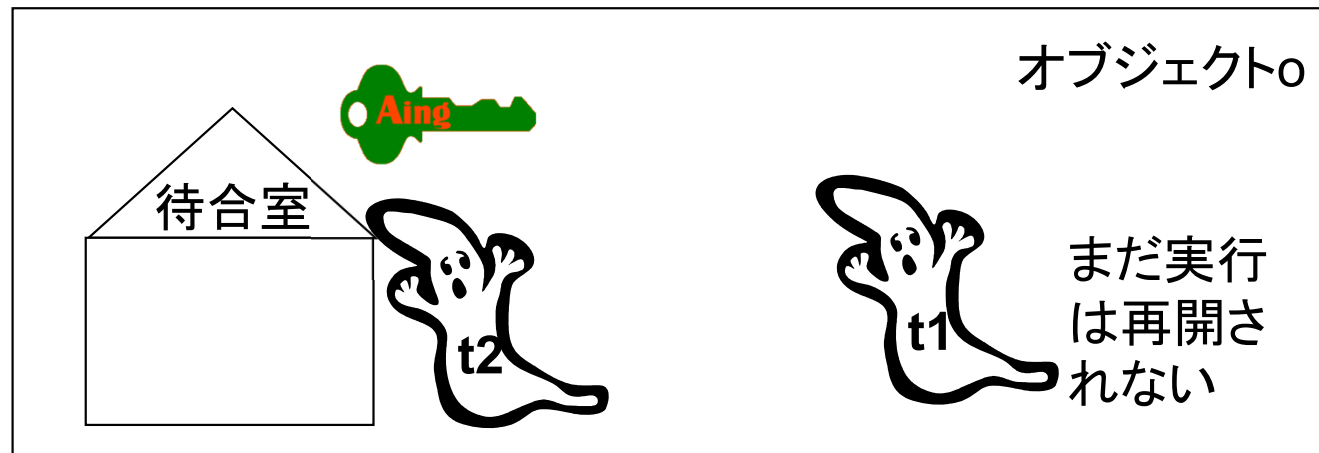
# スレッドの同期(Cont.)

- スレッドt1を起こすためには、別のスレッドt2がオブジェクトoのメソッドnotifyAllを呼び出す必要があり、そのためには、t2がoのロックを獲得する必要があるからである。



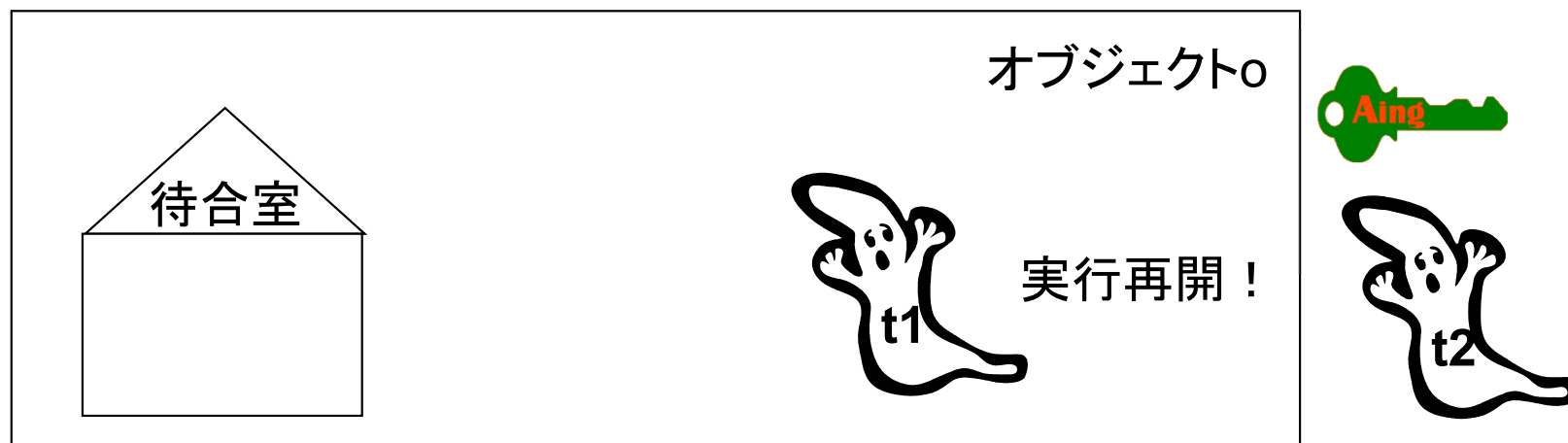
# スレッドの同期(Cont.)

- 待機集合中のスレッドt1が起こされても、そこはsynchronizedの部分の中である。従って、実行を再開するにはロックを獲得していなければならない。
- スレッドt2がnotify/notifyAllメソッドを実行した段階では、t2がロックを持っている。

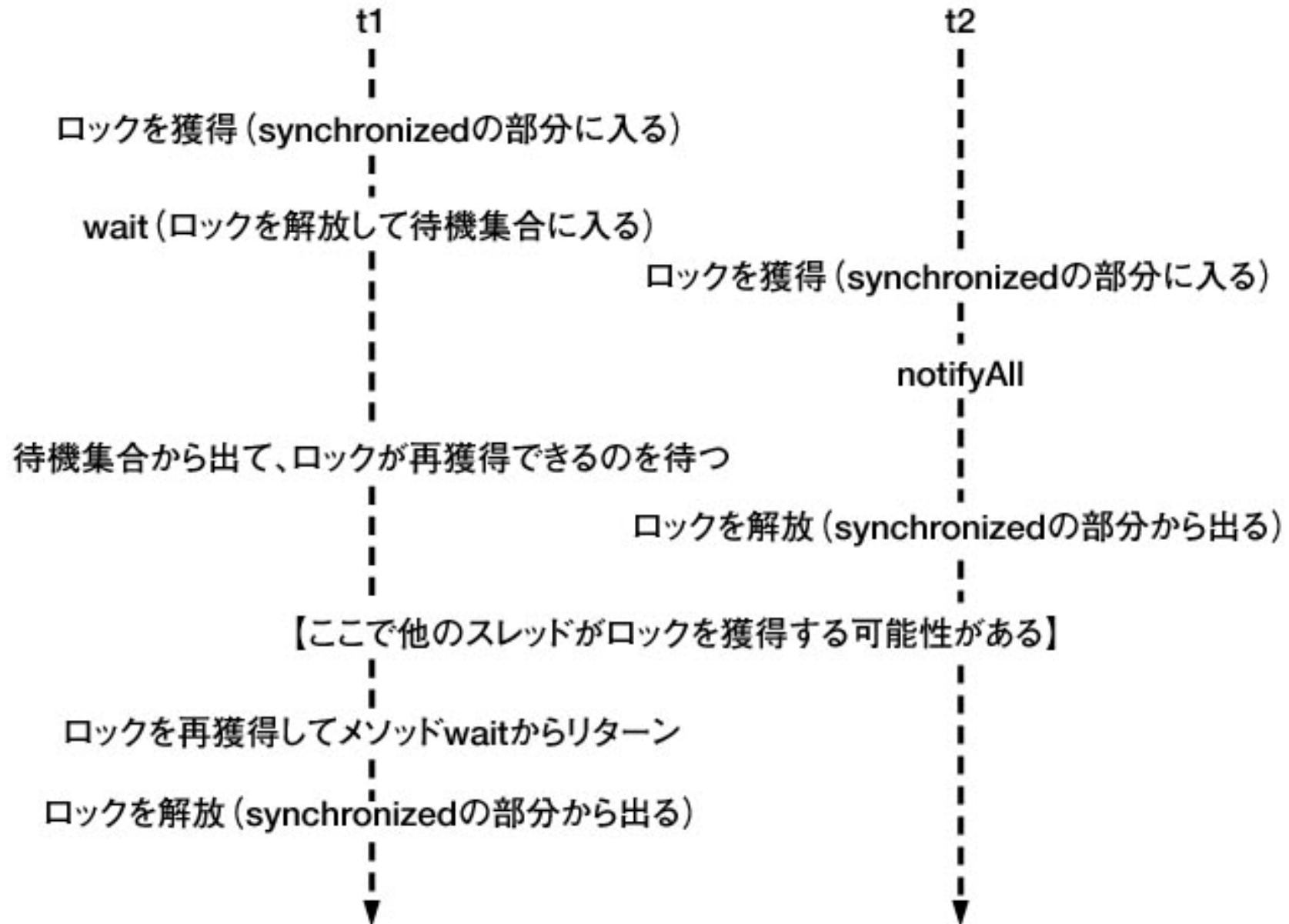


# スレッドの同期(Cont.)

- スレッドt2がsynchronizedの部分を抜けてロックを解放してくれないと、起こされたスレッドt1はロックを再獲得することはできない。
- ロックの再獲得を行うことで、はじめて実行可能な状態になる。
  - notifyやnotifyAllで起こされたスレッドは、すぐに実行されるわけではないことに注意。



# スレッドの同期(Cont.)



# スレッド間同期プログラム例

- Producerスレッド(生産者): 0から9までの整数を1つずつキューに入れる
- Consumerスレッド(消費者): キューに入っている整数を1つずつ取り出し, プリント

```
public class ProducerConsumer {  
    public static void main(String[] args) {  
        MyQueue q = new MyQueue(); //共有キュー  
        Producer p = new Producer(q);  
        Thread tp = new Thread(p); //Producerスレッド作成  
        tp.start(); //Producerスレッドスタート  
        Consumer c = new Consumer(q);  
        Thread tc = new Thread(c); //Consumerスレッド作成  
        tc.start(); //Consumerスレッドスタート  
    }  
}
```

# Producerクラス

```
class Producer implements Runnable { //Runnableインタフェースを実装
    private MyQueue queue;
    public Producer(MyQueue b) {
        queue = b;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            queue.putValue(i); //queueに整数iを入れる
            MyUtil.randomSleep(1000); //適当な時間待つ
        }
    }
}
```



# Consumerクラス

```
class Consumer implements Runnable { //Runnableインタフェースを実装
    private MyQueue queue;
    public Consumer(MyQueue b) {
        queue = b;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            int v = queue.getValue(); //queueから値を取り出す
            System.out.println(v); //それをプリント
            MyUtil.randomSleep(1000); //適当な時間待つ
        }
    }
}

class MyUtil {
    public static void randomSleep(long max) {
        long s = (long)(Math.random() * max);
        try { Thread.sleep(s); }
        catch (InterruptedException e) { }
    }
}
```

0.0 以上で、1.0 より小さい  
正の符号の付いた double  
値を返す

# キューの同期処理

```
class MyQueue {  
    private int value;  
    private boolean isEmpty = true;  
    public synchronized void putValue(int v) {  
        while (!isEmpty) { //queueに既に何か入っていたらスレッドを待たせる  
            try {  
                wait();  
            } catch (InterruptedException e) { }  
        }  
        notifyAll(); //queueに何も入っていなかったら他のスレッドに通知  
        isEmpty = false;  
        value = v; //queueに値を入れる  
    }  
    public synchronized int getValue() {  
        while (isEmpty) { //queueに何も入っていなかったらスレッドを待たせる  
            try {  
                wait();  
            } catch (InterruptedException e) { }  
        }  
        notifyAll(); //queueに既に何か入っていたら他のスレッドに通知  
        isEmpty = true;  
        return value; //queueの中身を返す  
    }  
}
```

# エスケープシーケンス

- Javaでは特殊な文字を表現するために、「¥」で始まるエスケープシーケンスが用意されている.

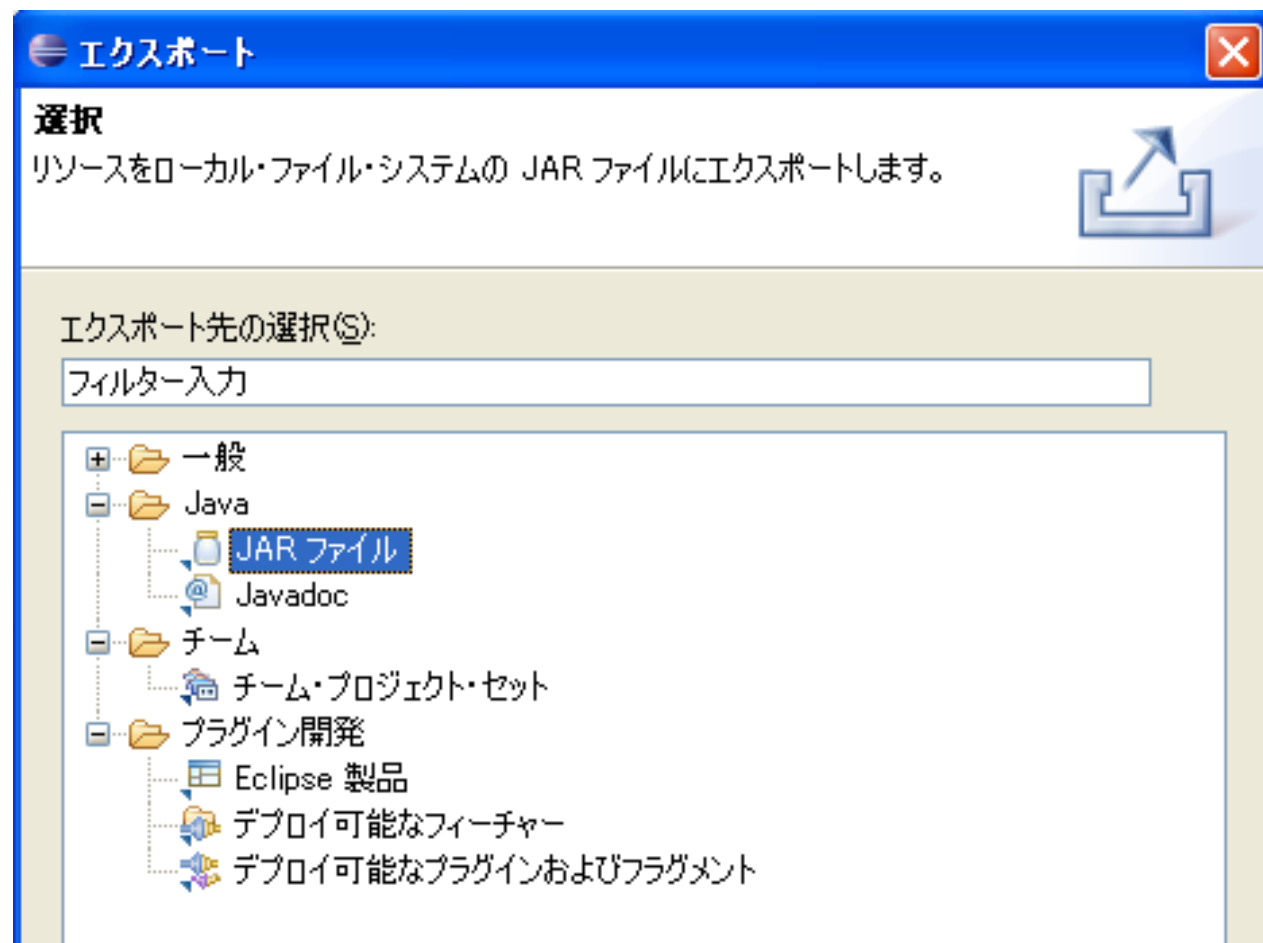
種別	説明	使用例
¥n	改行コード。	"Hello.¥n"
¥r	キャリッジリターン文字	"Hello.¥r¥n"
¥t	タブ文字。	"Name:¥tTanaka"
¥¥	バックスラッシュ。	"¥¥"
¥'	シングルクォーテーション (')	'¥'
¥"	ダブルクォーテーション (")	"¥"
¥uxxxx	Unicode xxxx の文字。	"¥u3042"

# jarファイル

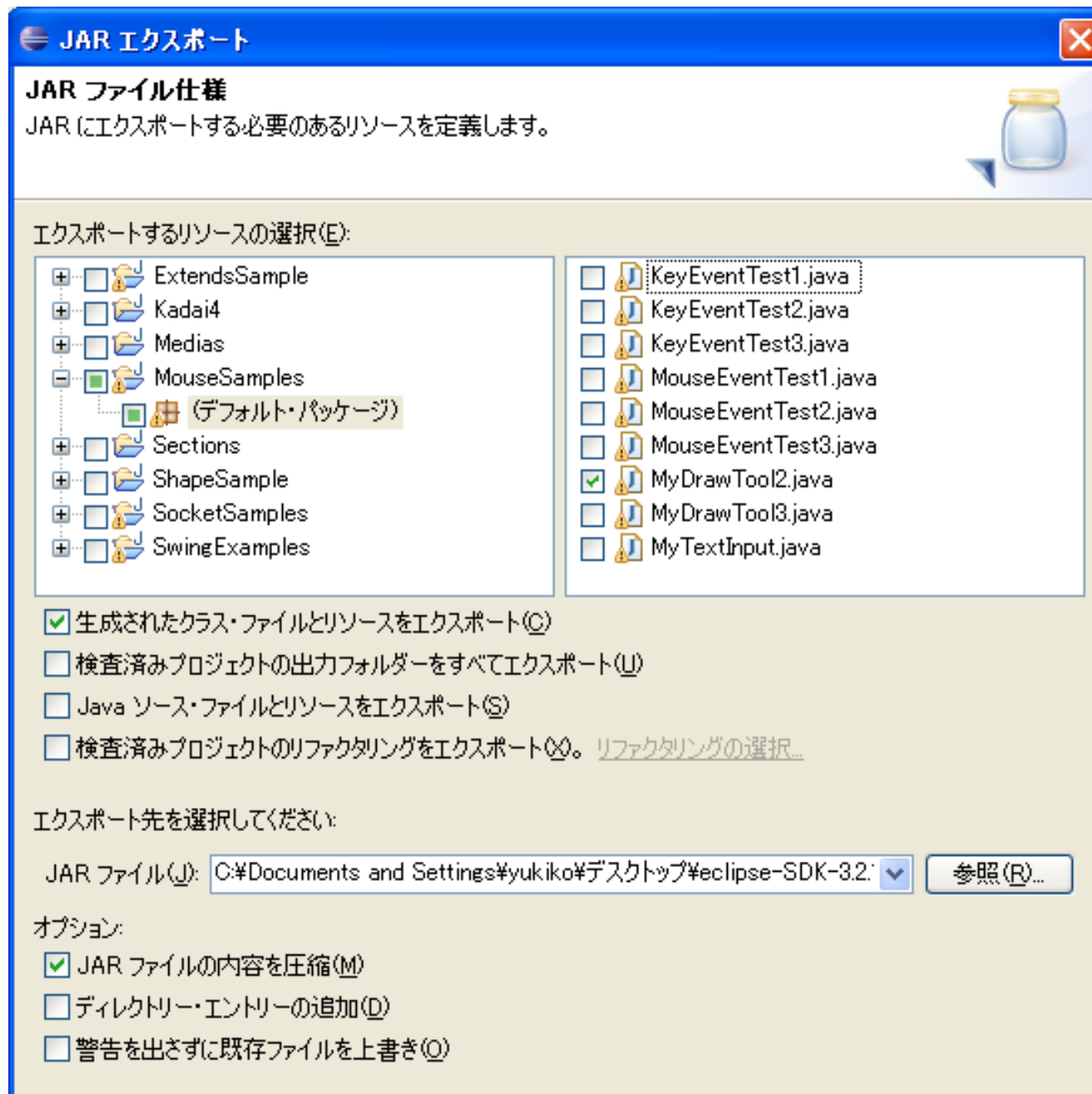
- 作成したプログラムを配布するには, jarファイルが便利である.
- javaの使い方を知らない人でも, ダブルクリックだけでjavaのプログラムを起動することができる.
- プロジェクトや複数のファイルをまとめて1つのjarファイルを生成できるので, ディレクトリの構成やフォルダ名が異なることによりプログラムが動かない, といった問題がおきない.
- ソースファイルを公開せずに, プログラムを配布できる.

# Eclipseを使ったjarファイルの作り方

- (1) Eclipseのファイルメニューから「エクスポート」を選択する.
- (2) エクスポートのダイアログからJavaの中の「JARファイル」を選択して「次へ」.



(3) JARファイル仕様の画面で、エクスポートするリソースを  
チェックマークで選択する。同時に、ファイルのエクスポート  
先(保存先)も指定しておく。



# Eclipseを使ったjarファイルの作り方

(4) JARパッケージオプションはそのまま

(5) JARマニフェスト仕様で、エントリーポイント(mainメソッドのあるクラス)を指定.

- マニフェストファイル

- JARファイルはzip形式なので、解凍ソフトにかけて中を覗いてみると、クラスファイルのほかに、META-INFというディレクトリが作られているのがわかる.
- META-INFディレクトリの中には、MANIFEST.MFというファイルがある.
- MANIFEST.MFはテキストファイルなので、エディタで開くことができる. この中にエントリーポイントとなるクラス名が指定されている.

# jarファイルの起動方法

- ダブルクリック
- > `java -jar jarファイル名.jar`