

Chapter 12. 実数と金利計算

12.1. 整数と実数の型の変換

実際の数値計算では、整数以外に実数を基本として計算することが多いでしょう。たとえば、金銭計算などの数値情報にしても、利率などを考えると実数を使わざるを得ません。これまでは、主に整数を基本として数値計算を考えてきましたが、ここでは実数を基本とした数値計算を考えましょう。まずは、整数と実数の間の変換から考えます。

★暗黙の型変換

実数が式に現れた場合は、Javaでは評価結果は、必ず実数（しかもdouble型）になります。次の例のように、実数で答えが欲しい場合は、演算の対象となる数値を実数表記（小数点をつける）にします。

例：	10/8.34	答えは実数
	10/8	整数の除算で、答えは整数の1
	10/8.0	実数の1.25が答え

変数への代入としては、整数の式を実数型の変数に代入するときは、そのまま何もする必要はありません。次の例のように、実数に変換されて代入されます。

例： **double** x = 10; // xには10.0が代入される

★明示的な型変換

逆に、実数を整数に直すような場合は、丸め（Round）を行わなければなりません。丸めとは小数部分を取り去る操作のことを指します。そのためには、明示的な型変換が必要となってきます。まず、式の評価結果を別の型の値に変換するための書式を紹介しましょう。

型を別の型に変換するためには、キャスト（cast）という式が用いられます。

▼キャストの書式

（変換したい型） 式

型によっては、変換できないものもあります。例えば、5という数はそのままでは文字列の"5"に変換することはできません。しかしながら、整数と実数の間ではキャストを使えばかなりうまく値を変換することができます。次の例は、幾つかの式を別の型に変換させてみたものです。

例：		
	(double) 5 / 2	// 結果は5.0 / 2となり2.5である。
	(int) 34.1	// 結果は、34となる
	(int) -45.5	// 結果は、-45となる
	(long) 45	// 結果は45Lとなる

例にありますように、実数から整数への変換は、切り捨てという形になります。これを Java では「0 に近くなるように丸める」と呼んでいて、整数部だけを取り出すことになります。他の言語では、truncate あるいは trunc 関数と呼ばれているものが、この操作にあたります。

実数の式を整数型の変数に代入するときは、このキャストを用いて型の強制変換を行なう必要があります。ここで、よく間違いがちなのが優先順位です。キャストは、単項演算子の一種で、掛け算や割り算などよりも高い優先順位を持ちます。そのため、次の例にありますように、括弧をうまくつけないと整数の式になってくれません。

```
例：   int    z = 10/8.34;           // これは間違い、エラーになる
        int    z = (int) 10/8.34;    // これも間違い、優先順位が違う
        int    z = (int) (10/8.34);  // これが正解
```

ただし、実数の式の評価結果が、整数で表せる範囲を超えた場合は、エラーになります。例えば、次のような実数を整数型にして、代入するとプログラムの実行時にエラーが発生します。

```
int    x = (int) 45.23e56;           // 45.23 × 1056は、整数では表しきれない
```

これは実行時のエラーに分類されます。後の章で出てきますが、実行時のエラーは例外と呼ばれています。

★実数の誤差と限界

たとえば、 $1/3.0 * 3.0$ は、通常は 1.0 になります。3.0 で割る操作を 100 回繰り返して、もう一度 100 回 3.0 倍する操作を繰り返しても、理論的には同じことで結果は 1.0 になるはずですが、それでは実際に次のようなプログラムの断片の部分を実行させてみましょう。

```
double x = 1.0;
for (int i=0; i<100; i++) { x=x/3.0; }
for (int i=0; i<100; i++) { x= x*3.0; }
System.out.println("result is "+x );
```

ところが、表示は次のようになりました。1.0 にはなっていません。どうしてでしょうか？

```
result is 1.0000000000000009
```

Java では内部的には、有限のデジタル情報として実数を表しています。そのために、本来アナログ情報である実数を表し切れない場合があるのです。本来のアナログ情報である実数とデジタルで表した場合の実数との差を誤差と呼んでいます。実数の計算には、誤差がつきものなのです。Java では、実数を 128bit の高精度で表していますので、従来のプログラミング言語で頻繁に起きていた誤差がかなり少なくなっています。しかし、上のように単純な割り算・掛け算も繰り返して行なわせると誤差が入り込んできます。

誤差の多かった時代には、なるべく誤差を少なくするような形で実数の計算が行なわれていました。誤差が発生しやすいときはどんなときでしょうか？経験的に、次のようなことがよく言われています。

「違うオーダー（大きさ）の数を足し算したり、引き算したりすると誤差が発生する」

たとえば、非常に大きな数と小さな数を足し合わせることを考えてみてください。10000 に 1 を足すときは、

10000であっても10001であっても、数の大きさ的には大きくは変わりません。同じように、有限のデジタル情報として実数が表されるとき、小さな差は丸められてしまいます。この丸めが誤差を発生する原因となっています。実数で計算を行なう場合は、このような大きさが著しく違う数を足したり、引いたりすることを避けなければなりません。

12.2. 実数関係のクラスライブラリ

■Mathクラスについて

Mathクラスには、数学上使う便利な関数がメソッドとして用意されています。「もう数学なんて嫌！」っていう人が多いかも知れません。しかし、いざというとき（ないと思っていると案外とあるでしょう）には必要となってきます（注1）。

メソッド	評価結果の型	意味
E	double	自然対数の底を表す
PI	double	円周率を表す
ceil(実数)	double	大きいか等しい一番小さな整数に丸める
floor(実数)	double	小さいか等しい一番大きな整数に丸める
rint(実数)	double	一番近い整数に丸める
round(実数)	int, long	実数を整数に四捨五入して丸める
acos(実数)	double	$\cos^{-1}$ を求める ($-\pi/2 \sim \pi/2$ の間の角度)
asin(実数)	double	$\sin^{-1}$ を求める ($-\pi/2 \sim \pi/2$ の間の角度)
atan(実数)	double	$\tan^{-1}$ を求める ($-\pi/2 \sim \pi/2$ の間の角度)
atan2(実数, 実数)	double	2つの実数をx、y座標とする極角度を求める
cos(角度)	double	cosを求める
sin(角度)	double	sinを求める
tan(角度)	double	tanを求める
abs(数)	double, float, long, int	絶対値を返してくる
exp(実数)	double	自然対数のべき乗を求める
log(実数)	double	自然対数を求める
pow(実数, 実数)	double	べき乗を求める
random()	double	0.0以上1.0未満の間で乱数を返してくる
sqrt(実数)	double	平方根を求める

実際にこれらの関数を使って簡単な数式、あるいは代入文を記述してみましょう。

★整数と実数との間の変換に使われる関数

```
double ceil = Math.ceil( 44.23 );           // ceil = 45.0
double floor = Math.floor( -44.23 );        // floor = -45.0
double closest = Math.rint( -44.23 );       // closest = - 44.0
int round = Math.round( -44.23 );           // round = - 44
```

整数への変換で注意しなければならないのは、ceilやfloorなどは、それぞれ負の数は、絶対値的には異なるような数に変換されるということです。また、roundだけは整数に変換されますが、他のものは関数の計算結果が実数のままであるということにも注意してください。また、roundは四捨五入ですので、下の例のような形になります。コメントとして書かれているのが、計算結果です。

<code>Math.ceil(23.45)</code>	<code>// 24.0</code>	<code>Math.ceil(- 23.45)</code>	<code>// - 23.0</code>
<code>Math.floor(23.45)</code>	<code>// 23.0</code>	<code>Math.floor(- 23.45)</code>	<code>// - 24.0</code>
<code>Math.round(23.45)</code>	<code>// 23</code>	<code>Math.round(- 23.45)</code>	<code>// - 23</code>
<code>Math.round(23.50)</code>	<code>// 24</code>	<code>Math.round(- 23.50)</code>	<code>// - 23</code>
<code>Math.round(23.55)</code>	<code>// 24</code>	<code>Math.round(- 23.55)</code>	<code>// - 24</code>
<code>(int) 23.45</code>	<code>// 23</code>	<code>(int) 23.45</code>	<code>// - 23</code>

ceilとfloorの関係、あるいはroundとfloorの関係を示すのに次のような公式が用いられることがあります。これで、各関数の関係を判断してください。

```
Math.ceil( x ) = - Math.floor( - x )
Math.round( x ) = (int) Math.floor( x + 0.5 )
```

★三角関数、逆三角関数

三角関数、逆三角関数の場合は、角度を指定するときはすべてradian体系（ π を基準とする体系）で行なわれます。通常の360度を使って角度を表す体系はdegree体系と呼ばれています。その間の変換は、次のようになります。なお、 π を指定するときは、JavaではMath.PIと記述します。

$\text{radian角度} = \text{degree角度} / 180 * \pi$ 例えば 90° は $1/2 \pi$

実際に、三角関数などを使った式を記述してみましょう。

```
double alpha = Math.acos( 0.33 );           // cos-1 0.33
double alpha = Math.atan2( 0.0, -2.33 );    // alpha = pi
double mycos = Math.cos( Math.PI / 3 );     // cos 60
```

★その他の関数

その他の関数としては、絶対値を求めるようなもの、自然対数関連のもの、べき乗や平方根などがあります。絶対値以外は、実数に対してしか用意されていないことに注意してください。

```
int abs = Math.abs( -44.23 );               // 44.23
double ep = Math.exp( Math.PI );           // eπ
double naturallog = Math.log( 1.34e5 );     // loge 1.34e5
double power = Math.pow( 4, 3 );            // 43
double rand = Math.random();               // 0.0 ≤ n < 1.0
double root = Math.sqrt( 2.0 );            // 1.414...
```

Math.randomは、乱数になっていて、0.0～1.0の間の適当な実数を発生して返してくれます。しかし、1.0という値は発生しませんので注意が必要です。たとえば、サイコロのように1から6までのどれかの整数を得たい場合は、次のように記述します。

```
int dice = (int)(Math.floor( Math.random() * 6 ) + 1); // 1～6までの乱数を発生
```

実は、これは乱数としてはあまり出来がよくありません。もう少しばらつかせたい場合は、かなり大きな数を掛けて、余りを取るという方式が良く採られます。

```
int betterdice = (int)(Math.random() * 534) % 6 + 1; // 534は適当な大きな数
```

(注1) 「いざというとき」は、近い将来思いがけずいきなりやってくるので注意しましょう。たとえば、この章の課題を来週までにやらないといけないなどという場合です。

■sin、cos関数を使ってみよう

★円を書いてみましょう

高校でやったかやらないかわかりませんが、三角関数のcosとsinを使ってx、y座標をなぞっていけば、円が書けるはず。以下の式でrは半径、tは中心の角度とします。

$$x = r * \cos(t)$$

$$y = r * \sin(t)$$

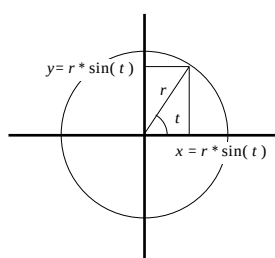


図12-1 円と三角関数（高校の教科書か？）

ところが、Mathクラスでは面倒なことに、角度はradian形式という角度体系を用いています。0から360度というのはdegree形式なので、それをradian形式に変換する必要があります。代表的な角度を以下に書き記してみましょう。

degree		radian	degree		radian
0	→	0	180	→	π
45	→	$\pi / 4$	270	→	$3\pi / 2$
90	→	$\pi / 2$	360	→	2π

こんな感じの対応を思い出して戴けましたか？公式を使って変換すると、実数のdouble型を用いて、Javaでは次のように式を表すことができます。たとえば、下の例では35度を変換して求めようとしています。

```
int degree = 35;
double radian = degree / 2 * 360.0 * Math.PI;
```

ここまでわかれば、あとはxおよびy座標を、radian角度と三角関数で求めることはできます。いま、半径 80 ドットぐらいを目安に考えましょう。しかしながら、アプレットのx座標y座標は、0からしか始まっていませんし、しかもy座標は逆になっています。そこで、中心を(100,100)の座標に置くように考えて、両座標にいつでも100を加えてやります。という訳で、これらを考慮すると、x座標、y座標は次のような形で求めることになります。

```
x = cos(radian) * 80 + 100;
y = -sin(radian) * 80 + 100;
```

なお、cosとsinは、Mathクラスのメソッドですし、最終的には実数で計算していたものを整数に戻してやらなければならないので、Java上では、次のように書くことになります。

```
x = (int) (Math.cos( radian ) * 80 + 100);
y = (int) (-Math.sin( radian ) * 80 + 100);
```

ところが、点を描くというメソッドがありませんので、線で円を描くことにします。つまり、折れ線で円を近似しようということになります。線を描くにはdrawLineメソッドを使うのですが、始点と終点の2つの点の座標が必要でした。2つの点をどのように決めましょうか？1つの点は上の計算で求めるとして、もう1つの点は前に計算した点の座標を覚えておくことにします。1つ前の点から、現在の点まで線を引くようにします。そこで座標を計算して線を描いた後に、

```
lastx = x;
lasty = y;
```

として、常に一つ前の点の座標を別の変数に覚えておけば、そこから、現在までの点まで線を引くことができます。

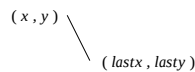


図12-2 前に求めた点から今回求めた点まで線を引く

最初はどうしましょうか？取り敢えず、上の式にradian=0を代入しますと(180, 100)になりますので、最初是从こから始めましょう。

```
lastx= 180;           // 100ずらしてあります。すなわち80+100
lasty= 100;           // 100ずらしてあります。すなわち0+100
```

という訳で、繰返しを使って角度は、0度から360度まで動かしてみて、線を描くようなアプレットを作ってみましょう。

```
import java.awt.*;
import java.applet.*;

public class MyCircle extends Applet {
    public void paint( Graphics gc ) {
        double radian;
        int x, y, lastx, lasty;

        gc.drawLine( 0, 100, 200, 100 );    // 横軸
        gc.drawLine( 100, 0, 100, 200 );    // 縦軸
        lastx= 180;
        lasty= 100;
        for (int i=0; i<=360; i++) {
            radian = i / 360.0 * 2 * Math.PI;
            x = (int) (Math.cos( radian ) * 80 + 100);
            y = (int) (-Math.sin( radian ) * 80 + 100);
            gc.drawLine( lastx, lasty, x, y );
            lastx = x;
```

```

        lasty = y;
    }
}
}

```

★リサージュ図形を描く

調子に乗ってきたところで、オシロスコープなんてものがありまして、あれって一体何ものかよくわからないけど、いろいろ測定する機械らしくて、いろんな図形がでるらしいのですが、理系以外の人にはあまり興味ないものかも知れません。そういうオシロスコープで、面白いのがリサージュ図形なのですが、これも理系以外の人にはあまりお馴染みありません。

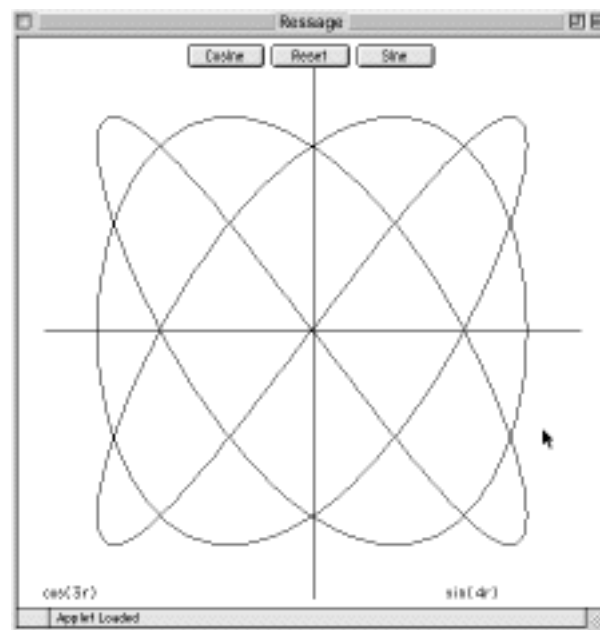


図12-3 リサージュ図形

この図形の特徴は、円を描くときの三角関数の角度の進み具合を変えてやればすぐ描けることです。円だったら $\cos$ で0から360度まで、 $\sin$ でも0から360度まで行儀良く変化させていきましたが、 $\cos$ で0から720度まで変えてしまったらどうなるでしょう？えっ、 $\cos$ だけ2回転もさせちゃったら、まずいでしょう？まあ、やってみましょう。

```
x = (int) (Math.cos( radian*2 ) * 80 + 100);    // 2倍2倍！
```

たとえば、上のようにx座標を求める式で、 $\cos$ 関数の角度を2倍するだけで、変な図形が出てきます。これをリサージュ図形と言うらしいのです。これボタンでどんどん押して何倍にもできるようにしちゃえ！って感じで次のアプレットを作ってみました。インスタンス変数 `zoom` に倍率をいれておき、ボタンが押されるたびに、どんどん倍率が+1されていくのが特徴です。

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Ressaga extends Applet implements ActionListener {
    Button button;

```

```

int      zoom =1;

public void init( ) {
    button = new Button( "Cosine" );
    button.addActionListener( this );
    add( button );
}

public void paint( Graphics gc ) {
    double radian;
    int x, y, lastx, lasty;

    gc.drawLine( 0, 100, 200, 100 ); gc.drawLine( 100, 0, 100, 200 );
    gc.drawString( "cos(" + zoom +"r)", 20, 190 );
    lastx= 180; lasty= 100;
    for (int i=0; i<=360; i++ ) {
        radian = i / 360.0 * 2 * Math.PI;
        x = (int) (Math.cos( radian*zoom ) * 80 + 100);
        y = (int) (- Math.sin( radian ) * 80 + 100);
        gc.drawLine( lastx, lasty, x, y );
        lastx = x; lasty = y;
    }
}

public void actionPerformed( ActionEvent e ) {
    zoom++;
    repaint();
}
}

```

12.3. 金利計算

社会系の学問でも実数を使った様々な計算が行なわれますが、ここでは一番身近な金利の計算を実数を用いて行なってみましょう。

■金利計算の基礎

★単利法と複利法

定期預金やローン計算などの利息を計算するときには、複利法を使います。元金に利率を掛けた結果が利息になりますが、単利法で利息を計算する場合と複利法で利息を計算する場合は、期間が複数に渡る場合には利息の額が異なってきます。

単利法…元金に期間数と利率を掛けたものが利息になります

複利法…期間毎に元金と利息の合計に利率が掛かったものが次の利息になります

それでは、例えば定期預金で10,000円を3年預けた場合を考えてみましょう。年利率1%とします。3年後に額

はいくらになるでしょうか？

単利法… $10000 + 10000 * 0.01 * 3 \rightarrow 10,300$ 円

複利法… $(10000 * (1 + 0.01)) * (1 + 0.01) * (1 + 0.01) \rightarrow 10,303$ 円

単利法と複利法では以上のように差がでてきます。利息の計算は通常は複利法で計算します。複利法の場合、貯める場合はより利息が大きくなります。逆にお金を借りる場合は、最初の利息が高くなります。また、数学的には複利法での計算は、等比数列の計算の応用となっています。

★年率、月率、日率

通常利息が使われるの年利率（年率あるいは年利）は、1年における利率を示しています。しかし、月で返済するようなローンや、日で返済するようなローンなどの場合は、1月の利率（月率）、あるいは1日の利率（日率）を求める必要があります。さらに、複利を使った定期預金でも、半年複利のものもありますし、ローンでもボーナス払い（年に2回と考えます）を別に用意しているものもあります。その場合には、半年の利率が利息計算に使われることになります。

半年の利率 → 年率を2で割ればよい

月率 → 年率を12で割ればよい（注1）

日率 → 年率365(366)で割ればよい

「金利」という言葉は通常は年率を示しています。注意しなければならないのは、消費者金融などで使われている実質年率と名目年率（約定年率）の違いです。

名目年率…預金金利や貸出金利のように契約で定められている年利率を指します。

実質年率…名目年率から物価上昇率（インフレ率）を差し引いた年利率です。

たとえば、名目年率が30%でも、その年の物価上昇率が5%であった場合には実質年率は25%になります。法律的には、年率は40%を上限としています。消費者金融の実質年率が28%と表記されていまして、名目年率はそれよりも数%高いのです。実生活でも注意したいものです。

★固定金利と変動金利

利率が一定の場合は、固定金利と呼びます。それに対して、変化するものを変動金利と呼びます。変動金利は、1年毎に利率が見直されるものと半年毎に利率が見直されるものとがあります。

固定金利 年利率が一定

変動金利 年利率が相場に応じて変わる

上限つき変動金利 変動金利だが、上限（これ以上高くない限度）はある

★定期預金の場合

それでは、定期預金を想定して、10000円を預けて、固定金利で、年利率が5.5%の場合（注2）、10年でいくらになるか、毎年表示してみましょう。毎年結果として求まる額を終価と呼びます。変数baseには元金、変数totalには終価が代入されます。また変数rateは、年利率が代入されています。totalとrateは、実数型の変数になっていることに注意してください。

```
public class Financial {
```

```

public static void main( String [] arg ) {
    int      base = 10000;                // 元金
    double   total = base, rate = 0.055;  // 終価と年利率

    for ( int year = 1; year <= 10; year ++ ) {
        total = total * (1+rate);
        System.out.println( year + "年後は、" + total + "円" );
    }
}

```

なお、等比数列の公式を用いれば、繰返しを使わなくとも、次の式で 10年後の終価を計算することができます。Math.powerの引数は両方とも実数なので、10年を10.0と記述しています。

```
double total = base * Math.pow( 1+rate, 10.0 );
```

※注1 正確には、月率は各月で日数が違うので日率をその月の日数分合計したのになります。ここで12で割っているのは概算で求める場合、あるいは月率を一定にする場合に使われる計算方法です。

※注2 こんなに高い預金金利は不況下の1999年現在は外国債ぐらいしかありません。

■年金計算

ここで言う「年金」とは老後のための積立金のことではありません。財務的には広く定期的にお金を積み立てたり、払ったりするもののことを年金と呼ぶのです。貯める場合と返済する場合で年金の呼び方が違います。

貯める場合の年金…積立金
返済する場合の年金…賦金

★期末払いと期首払い

積立金では、期首払い（それぞれの期の最初で年金を払う）場合と期末払い（それぞれの期の最後で年金を払う）場合では、利息が微妙に異なってきます。

期首払い…年金にも利息がつく
期末払い…年金には利息が付かない

たとえば、1万円の元金があって、年利率0.2%で毎年1000円の積立金であるような預金の場合に、期首払いでは1年後は11,022円ですが、期末払いでは11,020円となります。2円の差は、期首払いの場合に年金にも利息が付いたからです。賦金の場合は、期末払いがほとんどですが、その期に借りている額に対して利息が発生します。

★定期積立預金の場合

元金として10,000円を預けて、固定金利で年利率が5.5%の場合、毎年5000円を預けていくと、10年でどれくらいになるか終価を表示してみましょう。積立金は期末払いとします。

```

public class Saving {
    public static void main( String [] arg ) {

```

```

int      base = 10000, deposit = 5000;          // 元金と積立金
double   total = base, rate = 0.055;           // 終価と年利率

for ( int year = 1; year <= 10; year ++ ) {
    total = total * (1+rate) + deposit;
    System.out.println( year + "年後は、" + total + "円" );
}
}

```

なお、等比数列の和の公式を用いれば、次の式で10年後の終価を求めることができます。

```
double total = base * Math.pow( 1+rate, 10.0 ) + deposit * (Math.pow( 1+rate, 10.0 ) - 1) / rate;
```

■賦金（ふきん）計算

借入金を返済する方式には次のような3つの方式があります（注1）。それぞれの場合に、返済のイメージが異なります。

元金均等方式	月々決まった額の元金に利息を加えた金額を返済する方式
元利均等方式	借入元金と借入利息を合わせて月々、同じ金額を返済する方式
元金分割元金均等方式	借入元金が多い場合、元金を何年かごとに分割して、元利均等方式で返済する方式

★元利均等返済の場合

月々の返済額は一定になります。しかし最初の方の返済は、利息の部分の比率が非常に高く、利息分を返済していることになります。なかなか、元金（借入金）は小さくなりません。返済がある程度進むと元金分の割合も増えて利息は小さくなっていきますが、消費者金融などの年率の高いローンでは、月々の返済額を少なくすると利息ばかりを返済することになり、元金がほとんど返せなくなってしまうでしょう。返済総額は他の方式よりも大きくなります。多くのローンはこの返済方法を採用しています。

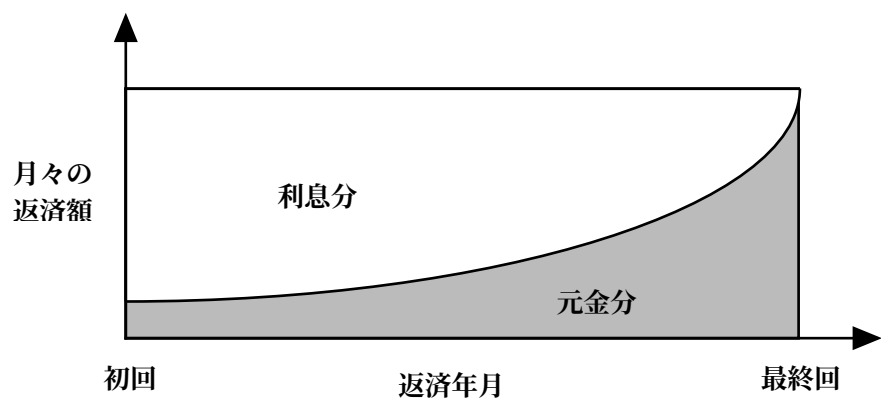


図12-4 元利均等返済

★元金均等返済の場合

元金と併せて、利息分も返済するので、最初の方の返済では月々の返済金額が非常に大きくなります。しかし、元金を毎回の返済で払っていきますので、利息は一定の比率で減少していきます。そのため、利息の総額

としては元利均等返済よりもだいぶ少なくなります。

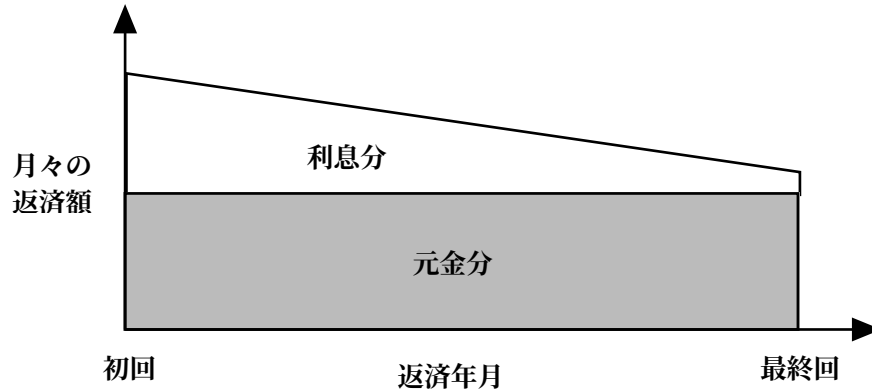


図12-5 元金均等返済

★元金分割（ステージ式）元金均等返済

借入金が多い場合、元金均等返済では最初の方の回の支払い金額が膨大になってしまいます。これを避けるために、借入金をいくつかのステージに分割して、各ステージごとで、元金均等返済を行なう方式です。借入金の規模が大きい中小企業への融資などに良く使われるとのこと。

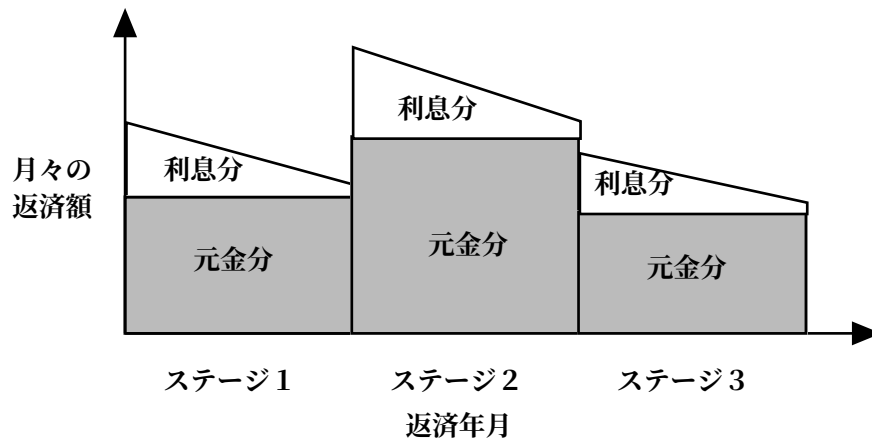


図12-5 元金均等返済

こんなこと関係ないと思っている方も多いでしょう。しかし、プログラミングばかり目を向けていて実際の社会の金利システムを勉強をしていないと、思わぬところで落とし穴に落ちることになります。

★ローン返済の場合（何カ月掛かるか？）

実際に賦金（ローン）の返済をプログラミングしてみましょう。たとえば、無謀にも300万円のBMWを買ってしまったとします。頭金が100万円として、固定金利で年利率が14.5%のときに、元利均等返済で月々50000円ずつ期末払いで払っていくと、何カ月で返せるか計算してみましょう。支払った月数と、残金を表示していきます。totalが毎回の返済残高、rateは12で割っていますので月率を表し、loanが月々の支払額を示しています。

```
public class Installment {
    public static void main( String [] arg ) {
        int    left = 3000000-1000000, loan = 50000; // 借入金と月々の支払額
        double total = left, rate = 0.145 / 12;      // 支払い後の残金と月の利率
```

```

        for ( int month = 1; total > 0 ; month ++ ) {
            total = total * (1+rate) - loan;
            System.out.println( month + "月後の残金、" + total + "円" );
        }
    }
}

```

★ローン返済の場合（月々いくら必要か？）

300万円のBMWを買おうと計画しました。幸い？頭金は50万円はあるとします。残りを5年ローン（つまり60回払い）で買いたいときに、元利均等返済で、利率は固定金利・複利で年利率 14.5%で、期末払いという条件で、一体月々幾ら払えば良いのでしょうか？外側の繰返しで、月々払うお金を10000円から始めて、100円単位で上げていき、払いきれかどうか試行しています。内側の繰返しでは60回分払ってめています。60回払った後の残高を表示していきます。もし、60回後に残高が0円よりも小さくなっていたら、月々その金額で払えば払い切れることになります。

```

public class Loan {
    public static void main( String [ ] arg ) {
        int    base = 3000000-500000;
        double total, rate = 0.145 / 12;

        for ( int loan = 10000 ;; loan += 100 ) {
            total = base;
            for ( int month = 1; month <= 60; month ++ ) {
                total = total * (1+rate) - loan;
            }
            System.out.println( "月々の支払:" + loan +
                               " 60回後の残金: " + total );
            if ( total < 0 ) { break; }
        }
    }
}

```

月々の支払額を求めるには等比数列の和の公式を使えば、上のような繰返しを使わなくとも、次のような式で求めることができます。また、利息の総額を支払総額から元金を引くことにより求めることができます。

```

double loan = base * Math.pow( 1+rate, 60.0 ) * rate / ( Math.pow( 1+rate, 60.0 ) - 1 );
double interest = loan * 60 - base;

```

※注1 ローンの返済の方法については、東京三菱銀行で店員の方にお聞きしました。また、返済のイメージ図に関しては、東海銀行のホームページを参考にさせて頂きました。

12.4. まとめと課題

■まとめ

★実数と整数との変換

- ・ 整数から実数へは自動的に変換される

- ・式に実数の項が一つでもあれば、式全体の評価は実数になる。
- ・整数から実数への変換は、キャストを用いる

★数学関数のクラスライブラリ

- ・小数の丸めには、さまざまな方式がある
- ・三角関数では、角度の表示にradian体系が用いられる
- ・乱数を発生させる関数が用意されている

★金利計算

- ・複利式と単利式がある
- ・年金には積立金と賦金がある
- ・支払いには元利均等支払い方式、元金均等支払い方式がある
- ・実質年率に注意しよう

■課題

12-1.

この章の例題を逐次実行してみなさい。金利計算の場合は、繰返しを使った例と公式を用いた例との両方を計算してみなさい。

12-2.

通常のサインカーブ、コサインカーブを描くようなアプレットを作成してみなさい。

12-3.

リサージュ図形を各アプレットを拡張して、Sineボタンも用意して、Sineの方の倍率も変えることができるように変更しなさい。また、倍率が大きくなりすぎた（たとえば 10 倍）になったときに、1に戻すように工夫しなさい。

12-4.

元金100,000円で、月々5,000円を入れる定期預金を開設して、ちょっと幸福になりました。しかし、100万円貯まるのは一体いつでしょう？元利均等返済の期末払いで、固定複利の年利率3%として、何カ月後に100万円を越すか計算するプログラムを作りなさい。クラス名は、Million。何カ月後になるか解答も求めなさい。

12-5.

Jaguar F-Typeという1,023万円のオープンカーを買うことにしました。無謀にも手持ちは0円です。元利均等返済の期末払いで、固定複利の年利率14.5%として、7年ローンとして、月々幾らか払えば良いか計算するプログラムを作りなさい。1円単位まで計算しなさい。クラス名は、Jaguarやっぱりでしょ。解答も求めなさい。



図12-7 Jaguar F-Type

ヒント：15万円ぐらいから、払いきれるかどうか試し始めるとよいでしょう。