ユーザインタフェース＃1 — 基本概念/基本的なUI操作

久野　靖*

2012.10.9

0　はじめに

本科目「ユーザインタフェース」は、人間とコンピュータのインタフェースを担うソフトウェアの基本的な考え方、枠組み、実現技術について学んで頂くことを目的としています。そのような科目の進め方としてはさまざまな可能性があると思いますが、せっかくの「ユーザインタフェース」なので、できるだけ「実際に作って動かす」ことを中心としたいと思っています。なので、各回とも講義と実習を両方取り入れる形で進めて行きます。言語はJavaを使用しますが、Javaが始めての人でも、何らかの言語でプログラムを書いたことがあれば問題ないように計画します。

また、各回とも実習の題材を元にした「課題」とReadingの「宿題」も用意しました。Readingについては読んで来てください。「課題」はできれば色々やって頂きたいですが、そこは時間が許す範囲内ということで。内容計画は次のように考えています。

- 第1回 ユーザインタフェースの基本概念 — ユーザインタフェースの概念と歴史、行動のモデル化
- 第2回 人間の行動モデル、Model Human Processor メニュー操作、時間計測、MHP、Fitzzの法則 —
- 第3回 打鍵レベルモデル、GUI — 打鍵レベルモデル、GUI部品、GUIの歴史
- 第4回 GUIの開発、ユーザインタフェースフレームワーク — プロトタイプ、モデルの分離、新しいインタフェース
- 第5回 ユーザインタフェースの研究分野 — ユーザインタフェースに関する論文紹介

このように、できれば最終回は論文紹介の回として、ユーザインタフェース関係の論文から自分の好きなものを1編選んで紹介することにしたいと思っています。成績は、各回の出席プラス論文紹介またはレポート(レポートは各回にある「課題」のうちからどれか1つを選んで書いて頂ければよいです)に基づいてつける、ということにします。そういうことでこれから5回、よろしくお願いします。

1　ユーザインタフェースの基礎概念

1.1　ユーザインタフェースとは

もともと、インタフェース(interface)とは、「何かと何かが接するところ」という意味なわけです。ですから、ユーザインタフェースは「コンピュータ」とそれを使っている「ユーザ(利用者)」とが接するところ、なわけです。これをコンピュータの側から見ると、「入出力装置」のうちで、人間とやりとりする(人間からの入力を受け取る/人間に出力を提示する)ものがユーザインタフェース(User Interface, UI)を構成しているわけです。

*経営システム科学専攻

ところで、「ユーザインタフェース」という用語の類語として、Human Computer Interaction (HCI) という言い方もあります。つまり、ユーザに限らず人間とやりとりする方が少し一般的ですね (ユーザでないけどコンピュータと関わる人間というのは何か考えてみると面白いです)。なお、これをひっくり返して CHI (Computer Human Interaction) というのは「カイ」という読みが語呂がいいので、SIGCHI、CHI'xx (カンファレンスの名前) などの形でよく使われます。

また、HCI といった場合には、「人間」が題材になっているだけあって、「認知科学的に」とか「人間の特性」とか「実験」など、心理学的な色彩が強いという感じもします。この科目が「ユーザインタフェース」なのは、自分がソフト屋であって、どっちかというと動くインタフェースを作る方が楽しくて好きだからそっちの雰囲気の強い用語を選んだというところがあります。でも原理は大切ですから、認知科学的な話もちゃんと出て来ますけれど。

1.2 ユーザインタフェースにおける課題

さて、コンピュータとユーザの接点であるユーザインタフェースにおいて、何が/どんなことが目標 (課題、テーマ) となるでしょうか? これには、コンピュータ側からの視点と、ユーザ側からの視点とがあると思われます。まずコンピュータ側から。

- コンピュータの処理速度/記憶容量 — CPU 能力やメモリが限られていた時代には、そのインタフェースのための処理時間やメモリ量が間に合うかどうか、ということは結構問題でした。今日では CPU が高速になり、メモリが廉価になったため、あまり問題ではなくなっていますが、実時間で高度なシステム (例: ゲームのグラフィクスなど) では問題ですし、スマートフォンなど CPU 能力が限られた中で使いやすいインタフェースを設計するためには CPU 能力とインタフェースのバランスは常に重要なテーマです。
- 実現の手間や複雑さ — 次々に新しいインタフェースが試みられ高度なものが産まれる中では、プログラミングの手間とか (自然言語でのやりとりみたいに) そもそも実現できるか、ということはより多く問題となってきています。できれば、手間は少なく、でも効果はあるインタフェースがよいわけですが…

次に、ユーザの側からの課題に進みましょう。

- 人間の操作所要時間 — もちろん、仕事が速く済むほど有難いに決まっています。
- 間違えにくさ/復帰しやすさ — 間違いがあると思ったことができませんから、間違えにくさは大切です。そして間違えることは必ず起きるので、そこから元に戻るのが容易なことも大切です。
- 人間の記憶負荷 — 沢山覚えている必要があると、速くてもしんどいかも知れません。
- 人間の学習/習熟負荷 — 最終的に効率がよくても、学ぶのが大変であれば、頻繁に使うようなものにしか採用できません。
- 人間の疲労 — 記憶や学習も含まれますが、たとえば細かい筋肉運動や視力を要求するものだと、すごく疲れるかも知れません。
- 分かりやすさ (視認性) — 何がどこにあるか、何をするにはどこを操作したらいいかがすんなり分かることは重要です。
- カスタマイズ — 自分のスタイルに合わせてインタフェースをカスタマイズできると使いやすくなるかも知れません。しかし、「何でも自由にカスタマイズしてください」とほうり出すのでも困ります。
- 親切さ/制御性 — 何かをやりたいと思った時にそれを手助けしてくれると楽にできます。かといって、押しつけられて本当にやりたいことの邪魔になるのは困るので、人間が制御を持っている、という感覚を維持することも大切です。

- 楽しさ — インタフェースの世界はどんどん、人間にとって「楽しい」方向に向かっています。

コンピュータの能力が増すにつれ、どんどんこちら側 (ユーザの視点での課題) の方が重要になっていることはお分かりでしょう。しかもとりわけ、最後の「楽しさ」は「機器の売れ行き」や「サイトの人気」につながるので、重要です。しかし「楽しい」とは何か、どうやったら楽しくなるのか、というのは永遠の課題です。

ちなにに現在では楽しさはユーザエクスペリエンス (User Experience, UX) という言葉で表されることが多いでしょうが、UX そのものはシステムが「何をさせてくれるか」まで含んでいるので UI よりはやや広い概念ということになります (もちろん UI も重要ですが)。

1.3 ユーザインタフェースの歴史

ユーザインタフェースの講義というと、やっぱりコンピュータの UI がどのように変化してきたかという話は不可欠なので、ひととおり説明して行きましょう。

ランプとスイッチ

コンピュータの最初期には、CPU は多数の部品からなる箱であり、そこから引き出されたスイッチやランプを使って直接レジスタの内容を見たりその値を設定したりできていました。現在でも、機械語を教えるのにはこの方法が最善だと思っている人は沢山います。

なぜ、こういうものがいいのでしょうか? それは、もともとコンピュータ内部のビットやレジスタやメモリ番地は「見えない」のですが、手でスイッチを操作し、ランプの点滅を見ることで、そこに具体的なイメージというか「実在感」が得られるからではないかと思います。

一方で、このような具体的なイメージが持てるかどうかが、コンピュータの操作を得意とするかどうかにつながっていた、という面もあります。今日でもたとえばプログラミングなどでは、その得手不得手はプログラムが実行することの内的イメージが持てるかどうかで左右されると思われます。

バッチシステム

バッチシステムとは、コンピュータの能力があがってきたけれど、依然として極めて高価だった時代のもので、パンチカードなどの形でたばねた「ジョブ」をセンターに提出すると、オペレータがその「ジョブ」を順次実行し、結果 (ラインプリンタ出力とか) をカードと一緒に返却してくれる、というものでした。

これも確かに「ユーザインタフェース」には違いありませんが、ターンアラウンドタイムが数時間とかも普通だったので、対話性という意味では最悪です。ただ、プログラマにとってはこの時代は「プログラムを紙の上でじっくり見る」ことになるためよかったという説もあります。結局、何が「よりよい」かの定義次第ですが…まあ、バッチシステムはコンピュータの利用効率に偏した方式であったことは確かです。

タイムシェアリング

タイムシェアリングシステム (Time-Sharing System, TSS) とは、高価なマシンに多数の端末が接続されていて、ユーザはそれらの端末の前に座って 1 つのシステムを「協同利用」する、というモデルです。その後、廉価なミニコンが出て来て、高価なマシンの代わりにミニコンを 1 人とか数人で使うのもこれに入るようになりました。

このインタフェースの画期的なところは、端末 (テレタイプライタか画面端末) の前に座ったユーザがコンピュータと対話的 (interactive) にやり取りできる、というところです。初期のテレタイプライタでは、QED とか teco のようなタイプライタエディタを使って「めくらで」ファイル内容を編集

していましたが、そして文字表示速度は毎秒 10 文字とかでしたが、それでもパンチカードや紙テープの打ち直し、切り貼りに比べたら圧倒的に便利でした。

その後、画面端末が普及しました。これは、決まった位置に決まった大きさの文字だけが表示できるという、現在から見たらひどく不便なものでしたが、それでも文字表示が高速であるというだけで、単にグラスタイプライタ (タイプライタの紙をガラス面にしただけのもの) としての使用でもずっと便利でした。PC(パソーナルコンピュータ) でも、初期の UI はコマンドを入力すると結果がそこに続いて表示される、というグラスタイプライタ方式が普通でした (MS は MS-DOS まで、Apple は Apple II まで)。

そしてほどなく、画面制御機能 (画面上の任意の位置にカーソルを動かしてそこの文字を書き換えられる) によって、現在のような「操作している近辺を画面に表示できる」エディタ (画面エディタ) が産まれました。今日でも「ターミナル (端末窓)」の中ではこのままの文化が生きています。

また、画面制御端末を使うと、「穴埋めフォーム」や「メニュー」なども表示できるため、キー操作によってフォームに記入したりメニューから選択するようなインタフェースが広く使われました。

グラフィック端末

上述のように、通常の画面端末は文字しか表示できなませんでしたが、製図など特殊目的のためには任意の図形が表示できる端末も非常に高価でしたが売られていました。たとえば Tektronix 401x という蓄積管 (電子ビームが当たると画面に線が描け、描いたものがずっと残って見える) を使った端末は広く使われていましたが、描いたものを消すときには「ぱっと全画面が光って全部消える」という方法しか手段が無く、それで動かすインタフェースを設計するのもひと苦労という感じでした (ですから、図形表示以外の場面では紙と同じインタフェースを使っていました)。

ビットマップ画面とポインティングデバイス

今日のような、「画面が多数の縦横に並んだ点」から構成されていて、各点の色をソフトで制御することで任意の絵が表示できる」画面 (最初は白か黒かを 1 ビットで表現したので Bitmap Display と呼ばれました) とマウス (ポインティングデバイス) を備えたのは、Xerox PARC (Palo Alto Research Center) で作られた ALTO というシステムが最初です。ちなみに ALTO 世界で最初の「パーソナルコンピューティング環境」であり、Smalltalk-80 言語、マルチウィンドウ、アイコン、WYSIWYG エディタなどをすべて始めて備えた画期的なものでした。

Xexox は ALTO を土台に研究用のワークステーションを複数開発し、最終的にオフィスワークステーション Star シリーズを販売しましたが、あまり成功はしませんでした (根強いファンはいましたが)。一方、ビットマップ画面とマウスを備えたワークステーションの概念は一気に広まり、Three Rivers PREQ、Apolo Domain、Sun などのシステムが販売されそれなりに売られました。初期のワークステーションはユーザインタフェースの実験場であり、さまざまなインタフェースが試みられました。今日の Unix の X Window にもその名残はあります (X そのものは特定の GUI を前提とせず、何でも実装できるように作ってあるなど)。

GUI と WIMP の時代

GUI(Grahical User Interface) を世の中に広めたのは何といっても Apple の Macintosh と、その後を追いかけた MS Windows ですが、これらはいずれも「複数のウィンドウが画面に開き」「デスクトップ (画面) にアイコンが並び」「メニューで機能を選択し」「それらをすべてポインティングデバイスで行う」という共通した枠組みを持っています。これを WIMP(Windows, Icons, Menus, Pointing device) インタフェースと呼びます。

また、インタフェースの実現に当たって窓、メニュー、ボタンなどの「GUI 部品」を標準化した形で提供し、それぞれ「ボタンは押せば何か起きる」みたいに見た目が工夫されているという点も特徴です。

WIMP インタフェースは「慣れない人でも見よう見まねで使える」という利点から広く普及しましたが、その反面でどのソフト、どのシステムでも似たような見え方であるとか、より新しいインタフェースのあり方が追求されなくなった、などのマイナスもあったかと思います。

今日〜これから

WIMP の時代もユーザインタフェース研究者はさまざまなインタフェースの研究をしてきたのですが、それが世に出ることはあまりありませんでした。しかし今日では、つぎのように新しいものが多数現れています。

- ゲーム機のインタフェース — ゲーム機はハードウェア込みでインタフェースをデザインできるので、Wii(リモコンを持って動かす) や Kinect(人間がさまざまな動きをする) のように新しい入力手段が使え、これまでに無かったインタフェースが作れます。
- モバイル機器 — タブレットやスマートフォンでは画面が限られている一方、タッチパネルを搭載しているので、画面をさまざまに触る (ドラグ、ジェスチャ、タップ) ことによるインタフェースが可能です。また、マルチタッチ機能を備えていれば、ズームや回転などこれまでにない操作も指示できます。
- 音声入力 — 文章やコマンドの音声入力はだいぶ前から使われて来ましたが、スマートフォンでは「電話」であってマイクとスピーカが搭載されていることから、Apple の Siri のような、人間が喋って指示する/コンピュータが喋って応答する、というインタフェースが今後普及する可能性があります。

この他にも研究レベルでは、実世界インタフェース (我々は普段、色々な実在のモノを見ながら生活しているが、そのモノ自身がインタフェースとして機能したり、モノの上にインタフェースが投影されたりする方式)、身体性インタフェースないし Augmented Human(サイボーグみたいに人間の身体の一部をインタフェースとして使用する)、など様々なものがあります。

ちなみに個人的には、自分はソフト屋なので、ソフトの中だけでできる世界に興味があり、「任意の絵が表示できる画面と、その画面上の好きなところを指させるポインティングデバイス」という枠組みだけでもまだまだ新しいことはできると考えていて、その可能性に興味があります。

1.4 モデルとメタファ

ここで、ユーザインタフェースを考えるときに重要な概念であるモデルとメタファについて取り上げておきます。モデルとは抽象化され一般化された何らかの枠組みであり、ソフトがあるモデルに従っていて、それがユーザにも分かっていれば、ユーザはソフトの動作を (そのモデルについて考えることで) 予測できるので、何をするにはどう操作したらよいかが分かりやすくなります。

このとき、モデルとしてユーザが既に持っているものを利用できると、モデルを学習してもらう手間が無くなるので有利です。たとえば、物理法則に従った物理世界そっくりにインタフェースを作る、などがこれに相当します。しかし、コンピュータでできるもの全てにそのようなユーザ手持ちのモデルがあるわけでもないという問題があります。そもそも、現実世界でそのままできないことをコンピュータにさせたいわけですから…

そこで、「そのもの」ではなくても、ユーザがよく知っているものやモデルにコンピュータの動作を「なぞらえる」ことで、そのよく知っているものから類推して動作を理解してもらえるようにする、という考え方が出てきます。これが「メタファ(比喩)」です。

メタファの代表的なものに「デスクトップメタファ」つまりコンピュータの画面をユーザの「机の上」になぞらえる、というものがあります。また、ボタンとかスライダーなどの GUI 部品もメタファの一部と言えるでしょう。

ただし、モデルにせよメタファにせよ、「完全にまねる」ことが目標ではないということも注意すべきです。たとえば、机の上がごちゃごちゃで何があるか分からなくなって困る人も沢山いますが、コンピュータの画面でも同じだったら不便ですね。コンピュータにはそれなりの「magic」が備わっていて、操作ひとつで「ものがさっと整列する」とか「欲しいものを言うと出て来る」のようなことができる、というのがユーザの求めていることなわけです。

1.5 ユーザインタフェースの評価方法

ユーザインタフェースをデザインして製作したら (もしかしたらデザインした段階までででも)、その評価が行いたいというのは当然です。評価基準としては、先に挙げた課題 (主にユーザ側の) を用いればいいわけですが、問題は評価値をどのようにして得るかです。

明らかな答えの 1 つは「主観的評価」(使って貰ってよいと思ったかどうかを調べる) で、これはある意味究極なのですが (最後に人間がいいと言わなかったら他の指標が良くてもだめ)、しかしそれだけではやはり困ります。

もう 1 つの明らかな答えは「実験」ですが、どうやって実験するか、というのも簡単な問題ではありません。たとえば、操作時間を目標とするなら、実際に操作してもらって時間を測るのでしょうが、「どんな作業を」やってもらうかによってその操作時間が何を測っているかが変わって来ます。つまり、本当に測りたいものを測っているか、という問題があるわけです。

そしてこれら以外にも、モデルにあてはめて計算する、みたいな評価方法もいくつかあります。これらにも含めて、評価の様々な側面については、実際に後の方で「やってもらいながら」考えて行きます。

2 ウィンドウシステムの入出力の枠組み

2.1 ウィンドウシステムとは

ウィンドウシステムという用語は最近ではあまり使われませんが、GUI の土台となる、「画面にさまざまなものを表示させたり、画面上をポインティングデバイスや指などで操作した入力を受け取る」機能を提供するミドルウェア、ライブラリ、フレームワークなどを指すと考えればよいでしょう。その基本的な機能・特性を挙げておきます。

- ビットマップ画面に対する表示を行う (多くの場合、窓の中への表示)。
- プログラムごとに「窓」を作り出させ、そこで対話を行わせてくれる。
- ユーザ操作による窓の切り替えが行え、入力もその生きている窓のプログラムに送られる。
- 入力はキーボード、マウス (ないし他のポインティングデバイス)、その他さまざまなものがあり得る。
- 入力はすべて「イベント」として送られる (イベントドリブン)。イベントにはデバイス、キーコード、ボタン番号、修飾キーなどさまざまな情報が付属。
- 出力は基本的には「任意のビットマップ出力」だが、図形を描いたり文字を描いたりするサポートがある。
- 出力はいつ描いてもいいのではなく、「いま描いて」と言われた時に描く必要がある。

2.2 Javaによるウィンドウプログラム

ではとりえあず、Javaによる窓の基本的な入出力をおこなうプログラムを見てみることにします。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Sample11 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  String str = "X";
  int xpos = 100, ypos = 100;
  Color col = Color.blue;
  public Sample11() {
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        int ch = evt.getKeyChar(); str = "" + (char)ch; repaint();
        System.out.println("key+: " + (int)(ch));
      }
      public void keyReleased(KeyEvent evt) {
        int ch = evt.getKeyChar();
        System.out.println("key-: " + (int)(ch));
      }
    });
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        requestFocus();
        xpos = evt.getX(); ypos = evt.getY(); repaint();
        System.out.println("mouse+");
      }
      public void mouseReleased(MouseEvent evt) {
        System.out.println("mouse-");
      }
    });
  }
  public void paintComponent(Graphics g) {
    g.setFont(fn); g.setColor(col); g.drawString(str, xpos, ypos);
  }
  public static void main(String args[]) {
    JFrame frame = new JFrame();
    frame.add(new Sample11());
    frame.setSize(400, 400);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

このプログラムの概要は次の通りです。

- このクラス Sample1 は、窓内の領域 (JPanel) を表すクラスを拡張した形として作っている。

- フォント、表示文字列、XY 座標、色はインスタンス変数としてどこからでも参照/書換えできるように保持する。
- コンストラクタ (初期設定メソッド) でキーボードイベント、マウスイベントに応答するコード (ハンドラ) を設定。
- キー押し下げイベントに対しては、その文字を文字列として覚えて再表示。押し下げ/離しいずれについても、そのイベントがあったことを起動した窓に出力。
- マウスボタン押しイベントに対しては、XY 座標を記録し再表示。押し/離しいずれについても、そのイベントがあったことを起動した窓に出力。
- メソッド `paintComponent()` は、画面の表示をせよという指示があったときに呼ばれる。その中で渡されて来た `Graphics` オブジェクトのメソッドを呼ぶことでさまざまな描画を行う。ここでは指定した色、フォントで、記憶しておいた XY 座標に記憶しておいた文字列を表示。
- `main()` はメインプログラムで、ここから実行を開始する。ここでは、窓を 1 つ作り、その中にこのクラスの領域をはめ込み、窓のサイズを設定し、窓を閉じる動作でプログラムも終わるように設定した後、窓を開く。

これを動かす操作は次の通り。

1. `cp /u1/kuno/work/Sample11.java Sample11.java`
2. `javac Sample11.java`
3. `java Sample11`
4. いろいろ操作してみる。
5. 終わりたい時は背景メニューの「DELETE」で窓をクリック。

演習 1 この例題プログラムをそのまま動かしてみなさい。動いたら、次のような変更を行ってみなさい。

a. マウスが離された時もそこの位置に動くようにする
b. 押し/離しで色が変るようにする
c. キーが押されたらその文字を文字列に追加する (「`str = str + (char)ch;`」でできるでしょう。ただし ch のコードが 32〜126 の時だけやる方がよい)。
d. BS キー (コード 8)、DEL キー (コード 127) の時は文字列の最後を切り捨てる

3 キーボード入力のインタフェース

3.1 キーボード入力の位置づけ

CUI (Character User Interface) ないし CLI (Command-Line Interface) は、コンピュータに対して文字列のコマンドを打ち込むとコンピュータからも文字列で応答があり、そのやりとりがグラスタイプライタな画面に表示されてスクロールしていく、という伝統的なユーザインタフェースを言います。今日では一般には GUI の方が普及していますが、次のような理由から今でも多く使われています。

- コマンドやパラメタを覚えている人が使うなら非常に高速
- 記録 (ロギング) の実装が非常に容易
- 実装が簡単でコード量が少ないので資源の限られた機器に有用
- 遠隔地から telnet 経由で使えるなど、環境に対する柔軟性が高い

そもそも、CUI はキーボードを入力装置として使うわけですが、キーボード自体にも次のような特性があります。

- 文章を入力するという目的では、現在でも一番高速 (将来的には音声入力の方が高速になるかも知れないが…)
- 文章に限らず、vi や Emacs などのエディタの操作に見られるように、「多数のスイッチを両手の 10 本の指で直接押す」という物理的なインタフェースは人間に非常に向いている。

ただし、日本語入力の場合にはかな漢字変換があるので、その分だけ直接性が薄まって不利という面もあります。このため、日本語入力でも直接打鍵ですべての文字 (漢字含む) を入力するという流儀もあります。

3.2 キーボードによる入力機能

では、キーボードという機器を「文字列を打ち込むための」入力装置として見るとどうでしょうか。裸の機器では文字コードが次々に来るだけですが、その上で次のようなさまざまな「機能」が実装されています。

- エコーバック — エコーバックがないと正しい文字が打てたかどうか不安なのは、パスワード入力のときを考えて見れば分かります。iOS などにある「最後の 1 文字だけ見える」というのはどれくらい有効でしょうか。また、「覗き見」を防ぐという意味ではどれくらい有効でしょうか。
- 打ち間違いの修正 — そして、文字列を打ち込むことを考えると「打ち間違い」は避けられないので、それからどうやって「復帰」するかは重要です。通常は、DEL キーを押すと最後に打ったものから逆順に消えて行きますが、これは自然な動作でしょうか? (何故?) 間違えた後、だいぶ打ってから気付いた場合は、エディタのように「カーソルを移動させて間違えた地点に行き、そこで挿入/削除により直す」方が合理的そうですが、それはそれで複雑です。むしろ、全部打ち直した方が速いかも知れません (どんな場合に? どうやったら判断できる?)
- 入力カーソル — 途中を修正するならカーソルは必要でしょうけれど、打つだけなら不要でしょうか。カーソルの点滅はありがたいのか、邪魔なのか、どちらでしょうか。
- 入力文字列自体に関する知識があると、より機能の大きいインタフェースが作れます。たとえば、途中まで打ったら残りが分かるような場合には、補完 (コンプリーション) することができます。では、勝手に補完するのでしょうか? 何かキーを押したら補完するのでしょうか? 補完できることはどうやって知らせるのでしょうか? また、途中まで補完できる場合はどうでしょうか。
- 補完だけでなく、入力されたものが「間違い」と分かるなら、それを訂正できるわけです。その場合、勝手に直されるとやはり問題でしょうから (例: 間違い例を説明する文書が作れない)、間違いを直しましょうと聞いて来ますか? それもうるさい気がします。たとえば入力文書に赤い波線がついたまま消えないとかは最近よくありますが…おかしい時は一瞬色が変わって知らせるけれど、徐々にその色は消えていく、とかどうでしょう。

3.3 基本的なキー入力機能の実現

以下に、基本的なキー入力機能のみを実現したプログラムを示します。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class Sample12 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  FontMetrics fm = null;
  String line = "";
  public Sample12() {
    setOpaque(false);
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) { requestFocus(); }
    });
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        char ch = evt.getKeyChar();
        if(ch >= ' ' && ch < 127) {
          line = line + ch; repaint();
        } else if(ch == 10 || ch == 13) { // CR, LF
          line = ""; repaint();
        } else if(ch == 8 || ch == 127) { // BS, DEL
          if(line.length() > 0) { line = line.substring(0, line.length()-1); }
          repaint();
        }
      }
    });
  }
  public void paintComponent(Graphics g) {
    g.setFont(fn); g.drawString(line, 20, 100);
    if(fm == null) { fm = g.getFontMetrics(); }
    int w = fm.stringWidth(line);
    g.fillPolygon(new int[]{20+w, 24+w, 28+w}, new int[]{105, 95, 105}, 3);
  }
  public static void main(String args[]) {
    JFrame frame = new JFrame();
    frame.add(new Sample12());
    frame.setSize(400, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

この例題プログラムは、先のプログラムを少し変更しで作られています。主要部分の説明を記します。

- 初期設定時に「`setOpaque(false);`」を呼ぶことで、画面が変更されることを明示 (重ね打ちとかにならないようにする)。
- マウスイベントでの動作は「`requestFocus();`」のみ。
- キーイベントの処理では次のようにする。
  1. 図形文字だけ `line` に連結する。
  2. CR、LF では入力完了と考え文字列を空に。
  3. BD、DEL では最後の文字を取り除く。

- 表示の処理の際、文字列が占める画面上の幅を求め、その最後のあたりに三角形 (カーソルのつもり) を描く。

演習 2 この例題をそのまま動かしなさい。動いたらのような変更を行ってみなさい。

a. Control-U を打つと入力が空になる。
b. BS、DEL で「最後の空白まで 1 単語ぶん」消えるようにする。
c. カーソルを Control-B、Control-F で任意位置に移動し、そこで挿入や削除ができるようにする。
d. 英語の月の名前を入力するものとして、補完機能をつける。

4 キー入力の時間計測

前節で、単なるキーボード 1 行入力といっても、さまざまな機能があり得ることがお分かり頂けたかと思います。さていよいよ、ユーザインタフェースの授業らしく、「所要時間」のことを考えて見ましょう。キー入力について、どのような「謎」があると思いますか? 少し挙げてみましょう。これらの「謎」については、考えたらとりあえず自分の予測をメモしておいておください。

- すべてのキーを打つのに掛かる時間は同じくらいなのか?
- 大文字を打つ (Shift-A) のは、小文字の倍掛かるか?
- 打っているものが英単語のときと、ローマ字の時で、時間の掛かり方は違うか?
- 日本語を見せられて、それを英語に直して打つのだとどうか?
- 1 文字打ち間違えた時、それを直して先を続けるまでに、どれくらい余分のペナルティがあるか?

さて、実際にこれらの「謎」を解明するには、実際に測って見ればよいですね。それも、ストップウォッチで測るとかだと大変ですし不正確なので、もちろんプログラムで測ります。先のプログラムを改造してキーボードの打鍵時刻が分かるようにしてみました。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Sample13 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  FontMetrics fm = null;
  String line = "";
  long time = System.currentTimeMillis();
  public Sample13() {
    setOpaque(false);
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) { requestFocus(); }
    });
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        long t = System.currentTimeMillis();
        char ch = evt.getKeyChar();
        if(ch >= ' ' && ch < 127) {
          line = line + ch; repaint();
```

```
        } else if(ch == 10 || ch == 13) { // CR, LF
          line = ""; repaint();
        } else if(ch == 8 || ch == 127) { // BS, DEL
          if(line.length() > 0) { line = line.substring(0, line.length()-1); }
          repaint();
        }
        if(ch > 32 && ch < 127) {
          System.out.println((t-time) + " : " + ch);
        } else {
          System.out.println((t-time) + " : " + (int)(ch));
        }
        time = t;
      }
    });
  }
  public void paintComponent(Graphics g) {
    g.setFont(fn); g.drawString(line, 20, 100);
    if(fm == null) { fm = g.getFontMetrics(); }
    int w = fm.stringWidth(line);
    g.fillPolygon(new int[]{20+w, 24+w, 28+w}, new int[]{105, 95, 105}, 3);
  }
  public static void main(String args[]) {
    JFrame frame = new JFrame();
    frame.add(new Sample13());
    frame.setSize(400, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

変更はごく少しで、まず冒頭で `System.currentTimeMillis()` を呼んで現在の時刻 (1970 年 1 月 1 日 0 時 0 分からのミリ秒単位) を取得します。次に、打鍵があったとき再度時刻を取得し、文字コード表示と一緒に出力します。

演習 3 あなたのキーボードの打鍵時間は平均どれくらいか、また、打鍵時間の分布はどんな風になっているか、検討しなさい。

演習 4 先に挙げた「謎」や、その他の自分が考えた「謎」について、実験を計画し、実施しなさい。2〜3 人でグループになり、それぞれの人の実験を交互に行いなさい (他の人の実験では被験者になる)。

たとえば、課題を提示して打ち込む実験なら、課題は紙に書いて見せてもいいです。グラフ用紙にタイムチャートを描いてみたりすると、より検討しやすいかも知れません。

5 本日のおまけ

5.1 自動問題提示

メインの内容は以上ですが、いろいろ実験プログラムを作ったので、せっかくだから示しておきます。まず最初に、問題 (英語の月名です) を自動的に提示する機能をつけてみました。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Sample14 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  FontMetrics fm = null;
  String[] months = { "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December" };
  String goal = "";
  String line = "";
  long time = System.currentTimeMillis();
  long goaltime = time;
  public Sample14() {
    setOpaque(false);
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) { requestFocus(); }
    });
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        long t = System.currentTimeMillis();
        char ch = evt.getKeyChar();
        if(ch >= ' ' && ch < 127) {
          line = line + ch; repaint();
        } else if(ch == 10 || ch == 13) { // CR, LF
          System.out.println(t - goaltime);
          line = goal = ""; repaint();
          Timer t1 = new Timer(
            1000+(int)(1000*Math.random()),
            new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                goal = months[(int)(Math.random()*12)]; repaint();
                long t1 = System.currentTimeMillis();
                System.out.println((t1-time)+" ! "+goal);
                goaltime = time = t1;
              }
            });
          t1.setRepeats(false); t1.start();
        } else if(ch == 8 || ch == 127) { // BS, DELETE
          if(line.length() > 0) { line = line.substring(0, line.length()-1); }
          repaint();
        }
        if(ch > 32 && ch < 127) {
          System.out.println((t-time) + " : " + ch);
        } else {
          System.out.println((t-time) + " : " + (int)(ch));
        }
```

```
        time = t;
      }
    });
  }
  public void paintComponent(Graphics g) {
    g.setFont(fn); g.drawString(line, 20, 100);
    g.drawString(goal, 20, 80);
    if(fm == null) { fm = g.getFontMetrics(); }
    int w = fm.stringWidth(line);
    g.fillPolygon(new int[]{20+w, 24+w, 28+w}, new int[]{105, 95, 105}, 3);
  }
  public static void main(String args[]) {
    JFrame frame = new JFrame();
    frame.add(new Sample14());
    frame.setSize(400, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

このプログラムでは、RET を打った時にタイマーをランダム時間指定で起動し、指定時間立つとタイマーから起動されるアクションで問題を提示するとともに、時刻を覚えています。そして、RET を打った時には覚えた時刻との差 (つまり全所要時間) も表示しています。

演習 5 自分でプログラムを動かし、タイムチャートを描いてどこでどのように時間が掛かっているか検討しなさい。その後、次のような変更を施したらどのような違いが現れると思うか、まず予測し、続いて (必要ならプログラムを手直しして) 実際にやってみなさい。

a. 回答として月名を 2 回くりかえし打つこととし、1 回目と 2 回目の時間がどう違うか調べる。
b. 回答を全部小文字で打ち込むこととし、時間の違いを調べる。
c. 問題の月名を日本語で、または数字で表示する。逆に月名は英語で表示され、日本語 (ローマ字) や数字で打ち込む。
d. 表示された「次の」月 (December の次は January) を打ち込む。

5.2 勝手に補完

さらに、答えが一意に決まる時は勝手に補完されるようにしてみました。1 文字打ち込むごとに、「ちょうどここまでで一意に決まる答えがあるか」調べ、あれば `line` をその答えで置き換えます。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Sample15 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  FontMetrics fm = null;
  String[] months = { "January", "February", "March", "April", "May", "June",
```

```
  "July", "August", "September", "October", "November", "December" };
String goal = "";
String line = "";
long time = System.currentTimeMillis();
long goaltime = time;
public Sample15() {
  setOpaque(false);
  addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent evt) { requestFocus(); }
  });
  addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent evt) {
      long t = System.currentTimeMillis();
      char ch = evt.getKeyChar();
      if(ch >= ' ' && ch < 127) {
        line = line + ch;
        int k = 0, c = 0;
        for(int i = 0; i < months.length; ++i) {
          if(months[i].toLowerCase().startsWith(line.toLowerCase())) {
            ++c; k = i;
          }
        }
        if(c == 1) { line = months[k]; }
        repaint();
      } else if(ch == 10 || ch == 13) { // CR, LF
        System.out.println(t - goaltime);
        line = goal = ""; repaint();
        Timer t1 = new Timer(
          1000+(int)(1000*Math.random()),
          new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
              goal = months[(int)(Math.random()*12)]; repaint();
              long t1 = System.currentTimeMillis();
              System.out.println((t1-time)+" ! "+goal);
              goaltime = time = t1;
            }
          });
        t1.setRepeats(false); t1.start();
      } else if(ch == 8 || ch == 127) { // BS, DEL
        if(line.length() > 0) { line = line.substring(0, line.length()-1); }
        repaint();
      }
      if(ch > 32 && ch < 127) {
        System.out.println((t-time) + " : " + ch);
      } else {
        System.out.println((t-time) + " : " + (int)(ch));
      }
      time = t;
```

```
      }
    });
  }
  public void paintComponent(Graphics g) {
    g.setFont(fn); g.drawString(line, 20, 100);
    g.drawString(goal, 20, 80);
    if(fm == null) { fm = g.getFontMetrics(); }
    int w = fm.stringWidth(line);
    g.fillPolygon(new int[]{20+w, 24+w, 28+w}, new int[]{105, 95, 105}, 3);
  }
  public static void main(String args[]) {
    JFrame frame = new JFrame();
    frame.add(new Sample15());
    frame.setSize(400, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

5.3 マウスで選択する場合と比較

だいぶ本題から外れていますが、マウスでも選択できるようにして、時間比較可能なようにしてみました。ぜひ、マウスとキーボード (+補完) で勝負してみてください。追加した機能は次の通りです。

- 12 個の黄色い領域を追加し、中に月名を表示した。
- マウスボタン押しの時、黄色い領域内だったらその月の入力として扱う。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Sample16 extends JPanel {
  Font fn = new Font("Courier", Font.PLAIN, 16);
  FontMetrics fm = null;
  String[] months = { "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December" };
  String goal = "";
  String line = "";
  long time = System.currentTimeMillis();
  long goaltime = time;
  public Sample16() {
    setOpaque(false);
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        long t = System.currentTimeMillis();
        char ch = evt.getKeyChar();
        if(ch >= ' ' && ch < 127) {
```

```
        line = line + ch;
        int k = 0, c = 0;
        for(int i = 0; i < months.length; ++i) {
          if(months[i].toLowerCase().startsWith(line.toLowerCase())) {
            ++c; k = i;
          }
        }
        if(c == 1) { line = months[k]; }
        repaint();
      } else if(ch == 10 || ch == 13) { // CR, LF
        System.out.println(t - goaltime);
        line = goal = ""; repaint();
        Timer t1 = new Timer(
          1000+(int)(1000*Math.random()), new MyAdapter());
        t1.setRepeats(false); t1.start();
      } else if(ch == 8 || ch == 127) { // BS, DEL
        if(line.length() > 0) { line = line.substring(0, line.length()-1); }
        repaint();
      }
      if(ch > 32 && ch < 127) {
        System.out.println((t-time) + " : " + ch);
      } else {
        System.out.println((t-time) + " : " + (int)(ch));
      }
      time = t;
    }
  });
  addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent evt) {
      requestFocus();
      long t = System.currentTimeMillis();
      int x = evt.getX(), y = evt.getY(), k = -1;
      for(int i = 0; i < months.length; ++i) {
        if(10+(i/6)*180 < x && x < 10+(i/6)*180 + 100 &&
           120+(i%6)*50 < y && y < 120+(i%6)*50 + 25) { k = i; }
      }
      if(k >= 0) {
        System.out.println((t-goaltime)+" ("+x+","+y+") "+ months[k]);
        line = goal = ""; repaint();
        Timer t1 = new Timer(
          1000+(int)(1000*Math.random()), new MyAdapter());
        t1.setRepeats(false); t1.start();
      }
    }
  });
}
class MyAdapter implements ActionListener {
  public void actionPerformed(ActionEvent evt) {
```

```
        goal = months[(int)(Math.random()*12)];
        repaint();
        long t1 = System.currentTimeMillis();
        System.out.println((t1-time)+" ! "+goal);
        goaltime = time = t1;
      }
    }
    public void paintComponent(Graphics g) {
      g.setFont(fn); g.drawString(line, 20, 100);
      g.drawString(goal, 20, 80);
      if(fm == null) { fm = g.getFontMetrics(); }
      int w = fm.stringWidth(line);
      g.fillPolygon(new int[]{20+w, 24+w, 28+w}, new int[]{105, 95, 105}, 3);
      for(int i = 0; i < months.length; ++i) {
        g.setColor(Color.yellow);
        g.fillRect(10+(i/6)*180, 120+(i%6)*50, 100, 25);
        g.setColor(Color.black);
        g.drawString(months[i], 20+(i/6)*180, 135+(i%6)*50);
      }
    }
    public static void main(String args[]) {
      JFrame frame = new JFrame();
      frame.add(new Sample16());
      frame.setSize(400, 400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
    }
  }
```

6 宿題

まず、Reading として次の 3 つのものを用意しました。次回までに読んで来て、Discussion できるようにしてください。お題は次回のお楽しみです。

- アラン・クーパー, 山形訳, コンピュータは、むずかしすぎて使えない!, 翔永社, 2000. (抜粋)
- D. A. ノーマン, 野島訳, 誰のためのデザイン?, 新曜社, 1990. (抜粋)
- 木村　泉, HCI は塩化水素か — 人とコンピュータの対話, bit, vol. 29, no. 4, pp. 16-20, 1997.

あと 1 つは、次の課題から 1 つ選んでやってみてください。結構大変なので、きちんとやったら最終レポートとして出す価値があると思います。

課題 1-1 「キーボードで何か打ち込む」作業の所要時間を予測するモデルを考えてみなさい。もちろん実験により確認してみること。

課題 1-2 本日の実習のうち、授業中にやって見られなかったものを 1 つ選び、やってみなさい。

課題 1-3 「月名が表示されたら、それをすばやく入力する」インタフェースでできるだけ「良い」(その定義は各自に任せます) ものを設計してみなさい。できれば作って実験してみるとよいでしょう。