平成 27 年度 プログラミング D Java 演習資料

担当教員: 石尾 隆　ishio@ist.osaka-u.ac.jp

演習課題

本演習では，オブジェクト指向プログラミングと Java についての理解を深めるために，演習完了時点で以下の 2 つの状態を達成することを目指す．

- 正しく動作する 1 つの Java プログラムを完成させている．
- 作成したプログラムを構成する様々な命令の意味を理解している．

本演習では，題材として「コンウェイのライフゲーム」を用いる．後述するゲームのルールと機能要件を満たしたプログラムを作成し，プログラムのソースコードとプログラムの動作を説明するレポートを提出することを演習課題とする．具体的な達成目標は次の通りである．

- 2015 年 11 月 12 日 (木) 23:59 までに，作成途中段階のプログラムを提出する．
- 2016 年 1 月 7 日 (木) 23:59 までに，完成したプログラムと，プログラムの内容について記載したレポートを提出する．

本演習では，前者を「中間提出」，後者を「Java 演習レポート」と呼んで区別する．中間提出は，プログラムの基本的な機能や提出方法について誤解がないことを確認するためのものであり，その時点の（コンパイルエラーのない）ソースコードのみを提出する．Java 演習レポートでは，完成したプログラムと，その動作を解説するレポートを提出する．いずれも大阪大学 CLE の「課題」として出題するので，12.2 節に記載してあるプログラムの提出用アーカイブの作成方法と，12.3 節のレポートの作成方法を参考にして提出を行うこと．

ライフゲームとは

コンウェイのライフゲームは，碁盤目状の格子からなる盤面にプレイヤーが「生物」を配置し，時間経過によって「生物」の生存状況が変化する様子を眺めるシミュレーションゲームの一種である．ライフゲームの各マス目のことを「セル」と呼ぶ．セルは「生きている」か「死んでいる」状態のどちらかをとる．プレイヤー（人間）は，最初に，盤面の各セルに対して好きなように「生きている」か「死んでいる」かの状態を設定する．この状態を「第 1 世代」として，盤面上のすべてのセルに対して「世代交代」のルールを適用することで，盤面を「第 2 世代」へと更新する．それ以降，繰り返し同じルールを適用していくことで，第 3 世代，第 4 世代というように盤面を更新していき，プレイヤーは機械的なルールによって生じる多様なパターンを眺めて楽しむことができる．初期状態（第 1 世代）の設定以外にはプレイヤーが関与することがないため，0 人ゲームと呼ばれている．

ライフゲームでは，ある特定の初期状態を与えると，特別な動きのパターンが観測できることが知られており，Wikipedia のライフゲームの項目 [1] に詳しい情報が掲載されている．Wikipedia 英語版，ドイツ語版など，解説の言語ごとにも異なるパターンが掲載されている．

ライフゲームのルールは次の通りである．

[1] http://ja.wikipedia.org/wiki/ライフゲーム

セルの生存条件：生きているセルの周囲 8 セルに 2 つまたは 3 つの生きているセルがある．ある「生きている」セルは，その周囲に 2 つあるいは 3 つの生きているセルがあるとき，次の世代でも「生きている」状態が続く．周囲に生きているセルが 1 つ以下の場合は孤立により，4 つ以上では過密により，そのセルは次の世代で「死んでいる」状態になる．

セルの誕生条件：死んでいるセルの周囲 8 セルに 3 つの生きているセルがある．ある「死んでいる」セルの周囲に 3 つの生きているセルがいるとき，次の世代，そのセルは「生きている」状態になる．この条件を満たさない死んでいるセルは，次の世代も死んだ状態のままである．

各セルからみて縦，横，斜めで隣接する 8 つのセルの状態を参照するが，盤面の端にあるセルについては，周囲の存在するセルだけを参照する．言い換えると，盤面の外側は常に「死んでいる」状態のセルによって囲まれているとみなす．

3 × 3 の盤面の 1 世代の更新例を次に示す．■を生きているセル，□を死んでいるセルとする．

```
■□□        ■■□
■■□   →    ■■□
□□■        □■□
```

- 左上の生きているセルは，セルから見て下と右下に計 2 つの生きているセルが存在するため，次の世代でも生存する．
- 中段左の生きているセルは，上と右に計 2 つの生きているセルが存在するため，次の世代でも生存する．
- 中央の生きているセルは，周囲に 3 つの生きているセルが存在するため，次の世代でも生存する．
- 上段中央，下段中央のセルは，現在死んでおり，周囲に 3 つの生きているセルが存在するため，次の世代では生きているセルが誕生する．
- 左下，右上，中段右のセルは，現在死んでおり，生きているセルは周囲に高々2 つしか存在しないため，次の世代も死んだままである．
- 右下のセルは，現在生きているが，周囲に 1 つしかセルがないため，次の世代では死亡する．

セルの生死はすべて直前の世代の盤面の状態によって定まる．将棋やチェスではプレイヤーによる盤面への操作を 1 手と表現するが，ライフゲームではプレイヤーによる手の選択がないため，機械的に世代すなわちゲーム内時間を進めていくことになる．

作成するプログラムの機能

作成するプログラムに持たせなくてはならない機能は，次の通りである．図 1 に，本演習課題に従って実装したライフゲームの画面例を示す．

GUI による盤面の表示．コマンドライン引数なしでプログラムを起動すると，ライフゲームの盤面を表示するウィンドウが開く．プログラム起動直後は，すべてのセルが死んでいる状態の盤面である．

盤面は，いわゆる碁盤目状の格子，すなわち小さな正方形の並びとなるように，大きな矩形（Rectangle; 長方形）の領域を垂直方向と水平方向の等間隔の線分によって区切ったものである．この境界線には 1 ピクセル幅の実線を用いること．

生きているセルと死んでいるセルを表現する 2 色を自由に選び，各正方形をそれぞれの状態に対応する色で塗りつぶす．盤面の外は，死んでいるセルと同色でも良いし，実行するオペレーティングシステ

ム標準のウィンドウの背景色などを用いても良い．すべてのセルが死んでいる状態で，盤面の格子がはっきり視認できる配色が望ましい．

動作環境は解像度 1024× 768 の画面を想定する．盤面のサイズ（セルの数）は 10×10 以上とする．縦横のセル数は異なっていても良く，セルの数は画面上で視認できる範囲で自由に決定して良い．

ウィンドウのサイズはユーザによって変更可能とする．サイズ変更を行ったとき，盤面の一部（たとえば端の境界線）が隠れてはならない．ウィンドウサイズは，縦横いずれも任意の最小サイズを設定してよい．

ウィンドウタイトルバーへのプログラム名の表示．ウィンドウのタイトルバーには,「Lifegame」という文字列を表示する．

マウスの左ボタンの押し下げによる盤面の編集．1 つのセルの境界線で囲まれた領域にマウスカーソルが存在している状態でマウスの左ボタンが押されたとき，そのセルの生死状態を切り替える．すなわち，カーソル位置にあるセルが生きていれば死んでいる状態に，死んでいれば生きている状態に変更する．マウス操作は，ボタンが押された時点でただちに画面に反映すること（ボタンが離されるまで待ってはいけない）．境界線そのものは，隣接するいずれかのセルの内部と考えてもよいし，どのセルの内部でもないと考えてもよい．

マウスの左ボタンのドラッグによる盤面の編集．マウスの左ボタンを押したままでのマウスカーソル移動（ドラッグ操作）により，盤面上のセルの生死状態を切り替える．マウスボタンを押すことで盤面の状態を変更したあと，ボタンを押したまま他のセルの領域にマウスカーソルが進入したとき，そのセルの状態を変更する．セルの切り替えは，領域への進入のたびに行う．たとえば，2 つのセルの領域を交互にカーソルが移動した場合は，2 度目の進入の時点でセルの生死の状態を元に戻す．盤面の外でボタンが押された状態になった後の進入に対しては，ドラッグと解釈してもよいし，しなくてもよい．

操作用ボタン ウィンドウは,「Next」「Undo」「New Game」の 3 つのボタンを持つ．ボタンをどこに配置するかは，盤面の内容がボタンによって隠されない限り，自由に決定してよい．図 1 の例では，ウィンドウ下端にボタンを並べている．

「Next」ボタンによる世代の更新．ボタンを押すごとに，盤面の状態を 1 世代進める．図 2 は,「グライダー」と呼ばれるパターンを描画した状態を示している.「Next」ボタンを押すと， 1 世代進んだ図 3 の状態に変化する．

「Undo」ボタンによる操作の巻き戻し．ボタンを押すごとに，盤面を直前の状態に巻き戻す．ここでの巻き戻しは，マウスボタンによって最後に変更されたセルの生死の状態や,「Next」ボタンによる世代の変更を巻き戻す．ドラッグ操作によるセルの変更は，1 セルごとの変更を巻き戻しても良いし，ドラッグ開始から終了までを 1 操作として巻き戻しても良い．連続で「Undo」ボタンが押された場合は，現在の盤面から最大で 32 操作前まで（それまでに操作が行われた回数と 32 回のどちらか少ないほうまで）さかのぼることができる．

「Undo」ボタンの自動的な有効・無効切り替え．Undo ボタンは実行開始直後は無効な状態[2] であり，一度でも盤面が編集されるなどすると有効な状態に切り替える．履歴がこれ以上ない状態まで巻き戻されると再び無効な状態に戻る．

「New Game」ボタンによる新しいウィンドウの表示．ボタンを押すごとに，新規の空の盤面を新しいウィンドウで開く．それぞれのゲームは同時に独立して実行可能であること．すべてのウィンドウが閉じら

[2]ボタンは表示されるが押せない状態のこと．オペレーティングシステムや GUI のシステムによって表現は異なるが，それぞれ固有の「無効」状態が定義されており，たとえば Windows 上ではボタンの文字列やボタン自体が灰色になる．

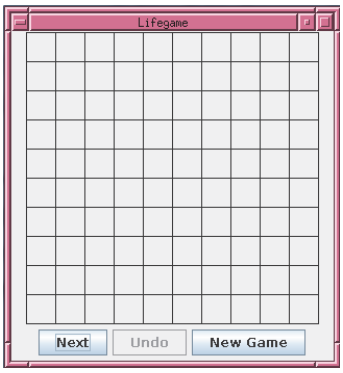

図 1: 起動直後の画面例．実行環境によってウィンドウの見た目は異なる．

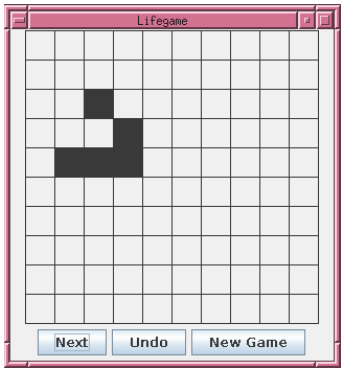

図 2: マウスによって生きているセルを盤面上に配置した状態．なお，このセル配置は「グライダー」と呼ばれるパターンである．

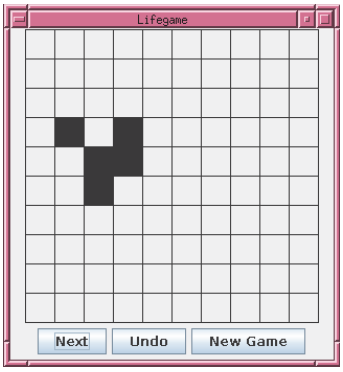

図 3: 「Next」ボタンを 1 回押し，図 2 から 1 世代進めた．この状態で「Undo」ボタンを押すと図 2 の状態に盤面を戻すことができる．

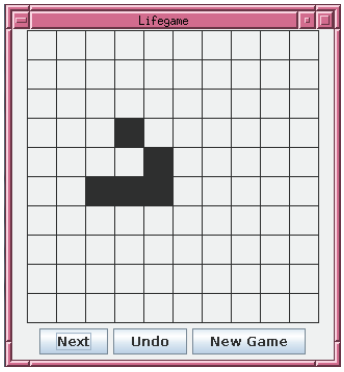

図 4: 「Next」ボタンをさらに 3 回押し，画面を更新した.「グライダー」は,「Next」ボタンを押すに従い，右下に向かって移動していく．

れたときプログラムを終了する．新たに起動されるゲームは，元のウィンドウと同一プロセスで実行すること（`fork` などを用いた別プロセスとしての起動は認めない）．

上記が必須機能であるが，プログラムとしてのライフゲームは，様々な機能拡張を加えることで，ユーザにとっての使い勝手を改善できる．もし時間に余裕があれば，機能拡張に取り組むこと．実装された追加機能について，独創性や技術的難易度をもとに評価を行う．

GUI に関する用語

計算機で使用するディスプレイには滑らかな画像を表示することができるが，実際には非常に小さな点**ピクセル** (pixel) の集合として管理されており，ディスプレイは解像度によって示される個数だけピクセルを二次元に並べたものだと考えることができる．たとえばディスプレイの解像度が 1024 × 768 であるということは，ピクセルが水平方向に 1024 個，垂直方向に 768 個並んでいるとみなすということである．

画面上に表示されるすべての点や線は，少なくとも縦横 1 ピクセルの大きさを持つ．たとえば図 5 に示す阪大ロゴの PNG 画像データの中央上部を拡大していくと，図 6 のような画像が得られる．図 6 における小さな四角形の領域が，元の画像における 1 つ 1 つのピクセルに対応する．画面上で図形を滑らかに見せるため，境界部分に薄い青色のピクセルを使用していることが分かる．

図 5: 阪大ロゴの PNG 画像

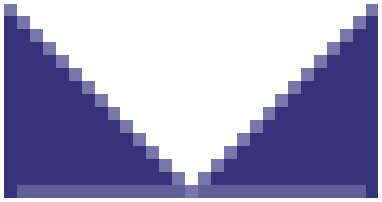

図 6: 阪大ロゴ中央上部を大きく拡大した画像

ディスプレイ上の各ピクセルは，**座標**によって識別される．本演習で使用する Java の GUI プログラミングでは，ディスプレイの左上を原点，右方向を X 軸，下方向を Y 軸とする．さらに，ウィンドウ内部の表示領域ごとに，それぞれの領域の左上端を原点とした相対座標を用いる．

マウスとは，ディスプレイ上の特定のピクセルを選択するデバイスである．マウスカーソルの先端はただ 1 つのピクセルに対応し，X 座標と Y 座標の組をプログラムに提供する．ライフゲームにおいては，セルを境界線によって区切るが，境界線自体も 1 ピクセルの幅を持つため，カーソルが境界線の上にあるという状態がありうることに注意が必要である．

プログラムがウィンドウにその内容を表示することを，本資料では**描画**と表記する．描画内容は通常，実行環境によって保持され，たとえばウィンドウを移動したとしても，ウィンドウの内容はそのまま表示され続ける．ウィンドウのサイズ変更が行われたり，ウィンドウが他のウィンドウに隠されて再度表示された場合には，プログラムはウィンドウの内容を**再描画**する．プログラムが再描画を行わない場合，ウィンドウの内容は失われる．この現象は，たとえば Windows においては，動作の遅いプログラムや応答しないプログラムのウィンドウの内容が真っ白になるという形で視認できる．

実装上のルール

プログラムの作成にあたっては，以下の規約を満たすこと．**規約を満たさないプログラムに対しては，レポートの再提出を求めることがある．**

プロジェクト名： `lifegame` と学籍番号を連結したものを使用すること．

パッケージ名： `lifegame` パッケージとそのサブパッケージにすべてのクラスを配置すること．

main メソッド： `lifegame` パッケージの `Main` クラスからプログラムを起動できるようにすること．

ソースコードの再利用に関する制限： 学習という目的の都合上，自作したソースコードと Java の標準ライブラリとを組み合わせて動作するものであること．機能拡張を行うために標準添付でないライブラリを使いたい場合は，教員まで申し出て許可を得ること．動作環境に制限があるものや，他者が作ったライフゲームそのものの再利用は許可できない．他人のプログラムの一部を流用した疑いがあるプログラムの提出に対しては，単位認定を行わない場合がある．

参考書について

Java は広く普及したプログラミング言語のひとつであり，インターネット上にライブラリの使用例など，数多くのサンプルコードや解説記事が公開されている．文法についてはオンラインにある言語仕様[3]や，「Java の道」[4]が便利である．Java の各クラスが持つ API の定義が知りたい場合は，オンラインの API 仕様[5]を

[3] `http://docs.oracle.com/javase/specs/jls/se6/html/j3TOC.html`
[4] `http://www.javaroad.jp/index_basic.htm`
[5] `http://docs.oracle.com/javase/jp/6/api/`

参照すること．GUI 関連のクラスの使い方などは，それらの名前で検索するだけでも，いくつかの記事を発見することができるので，API の一覧と合わせて参照すると良い．

書籍では，Joseph O'Neil「独習 Java 第 4 版」（翔泳社，2008）が，Java について詳しく知りたい場合に役立つ．例年の授業を通じて得られた意見によると，挿絵や比喩などの余計な装飾がないことから，既にプログラミング経験を積んできた人間には読みやすいという評価を得ているようである．ただし，Java では C 言語になかった新しい概念が多数，新しい用語として登場するため，理解できるところから順に読んでいく努力が必要である．

Java 特有のコードの書き方について知識を深めたい場合は，Joshua Bloch「Effective Java 第 2 版」（ピアソン・エディケーション，2008）が役立つ．また，Java の言語仕様を十分に理解した自信があるなら，Joshua Bloch，Neal Gafter「Java Puzzlers - 罠，落とし穴，コーナーケース」（ピアソン・エデュケーション，2008）に書かれたパズルを楽しんでもよい．

オブジェクト指向プログラミングの概念そのものや意義については，平澤 章「オブジェクト指向でなぜ作るのか 第 2 版」（日経 BP 社，2011）が詳しく解説している．

演習で使用する開発環境

本演習では，Java SE 6 (JDK 1.6) と統合開発環境 Eclipse を使用する．Eclipse のプロジェクトデータをエクスポートしたアーカイブファイルが提出物になる．現時点で最新である Java 8 で導入された記法を使用すると，演習室の環境ではコンパイルできなくなる．使用する Java 仕様のバージョン番号は Eclipse 側で指定できるので，授業では特に解説しない．

Eclipse プロジェクトの公式サイト [6] では様々なバリエーションが公開されているが，演習室に導入されているのは “Eclipse Classic” に属する Eclipse SDK 3.7.1 である．Eclipse のバージョンが大きく異なると，ファイルのやり取りの際に問題が生じる場合がある．

本資料のここからの表記について

本演習の目標は，ライフゲームのプログラムを完成させることである．本資料では，これ以降，プログラムの機能を中間ゴールに分けて段階的にプログラムを作成していく方法を紹介する．本資料に書かれた手順は，プログラムの設計を事前に整えてから一気に完成させる場合に比べると，少しずつ動作を確認できるという利点があるかわり，余分な作業（後から不要になるような処理の作成）も含まれている．プログラムの設計は，本来自由度が大きいものなので，この手順に従うことは必須ではない．プログラムを完全に自作する場合は，12 章の提出物に関する条件等だけを確認しておくこと．

本資料では，ライフゲームのプログラムを図 7 に示すような構造で作成する．この図において，楕円形の頂点は自作する主要なクラス，矩形の頂点はライブラリのクラスである．実線の矢印でクラス間の関係を表現している．主要なクラスの役割は次の通りである．

- `Main` は，プログラム起動時に実行される `main` メソッドを持つ．GUI の部品など，実行に必要な準備を整える役割を担う．
- `BoardModel` は，ライフゲームの進行に必要な「盤面」の状態（各セルの生死）を管理する．盤面の操作をメソッドとして提供し，盤面の状態が変化したときは，`BoardView` などに通知を行う．
- `BoardView` は，盤面の状態を画面に表示し，現在の表示状況を管理する．また，マウス操作を盤面の操作として解釈し，`BoardModel` に指示を行う．

[6] `http://eclipse.org/`

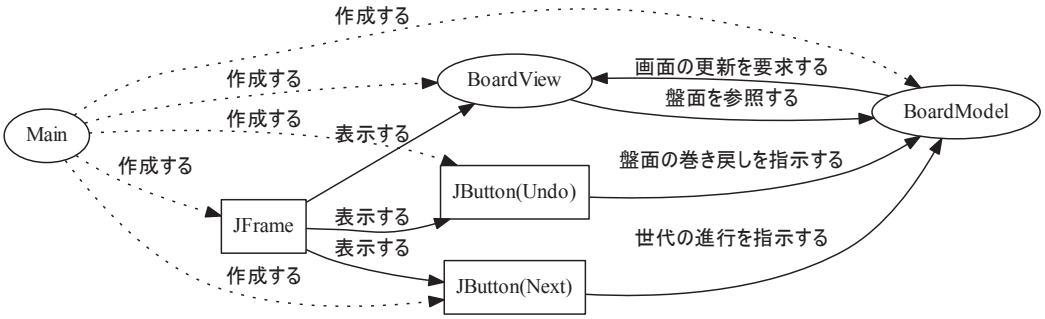

図 7: 作成するプログラムの大まかな構造

`Board` は盤面のことを，`Model` と `View` という単語はそれぞれ「データそのもの」と「画面上での見た目」という意味合いを持つ．これらの名前は Model-View-Controller モデルという GUI プログラムの一般的な設計法に由来する．

本資料の 1 章は，Eclipse の操作方法について演習時間中に逐次的に読んで作業，学習を進めるための手引書となっている．2 章から 6 章にかけて盤面の状態を管理する `BoardModel` クラスを作成し，7 章から 10 章にかけて GUI を構築していくことになるが，プログラムの書き方に自由度がないところについては機械的な手順をほぼそのまま示している．プログラムの内容を自分で考えて記述する必要がある場面と，手順的な場面が混在しているところも多いので，授業時間外に一通り目を通して，自分で考える必要がある点を洗い出しておくことを推奨する．

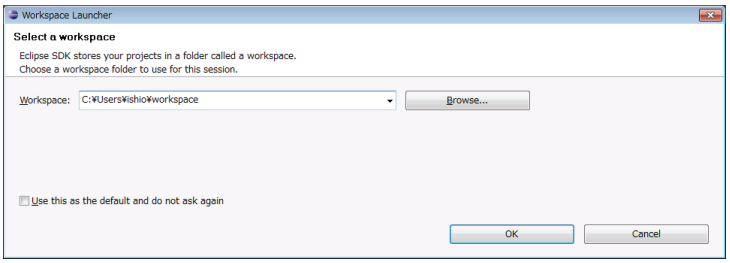

図 8: Eclipse 起動直後に表示されるダイアログ．指定ディレクトリにソースコード等が保存される．

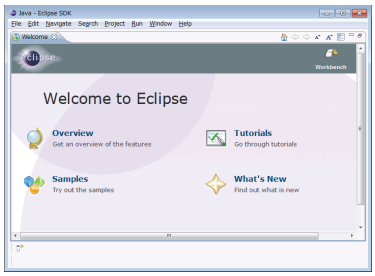

図 9: 「ようこそ」画面．演習室では表示が異なる可能性がある．

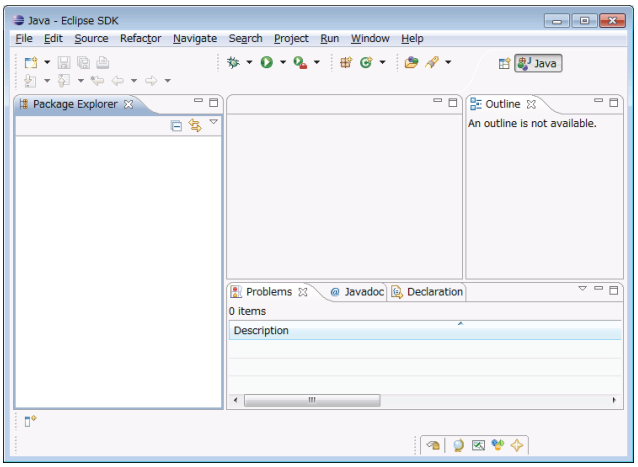

図 10: Eclipse ワークベンチの初期状態．様々な「ビュー」を格納したウィンドウである．

1 開発環境の準備

最初の準備として，Eclipse を起動し，作業空間としてのプロジェクトを作成せよ．また，ファイルの作成，保存とコンパイル，実行までの手順を確認せよ．

1.1 Eclipse を起動する

Eclipse を起動せよ．演習室の FreeBSD 上で Eclipse を起動するコマンドは `eclipse` である．多くのソフトウェアと同様に新しいウィンドウが開くので，Eclipse を起動したまま，シェルで他のコマンドを実行する予定がある場合は，`eclipse &` で起動する．

Eclipse を起動すると，まず図 8 のようなワークスペースの選択ダイアログが表示される．ここで選択したディレクトリに Eclipse が生成するファイルを保存することになる．通常はホームディレクトリの下に `workspace` という名前のディレクトリが作られるので，この名前で問題がなければ，そのまま [OK] を押して進む．今後，ディレクトリを変更するつもりがなければ，「この選択をデフォルトとして使用し，二度と問い合わせない (Use this as the default and do not ask again)」という項目にチェックをすることで，このダイアログを飛ばすことができる．

初回起動時は，「ようこそ (Welcome)」画面が表示される．Windows では図 9 のような画面が表示される（演習室の環境では文字列だけの表示になることもある）．「ワークベンチへ（Workbench あるいは Go to the Workbench）」という表記をクリックすると「ようこそ」画面は終了し，図 10 のようなウィンドウに変化

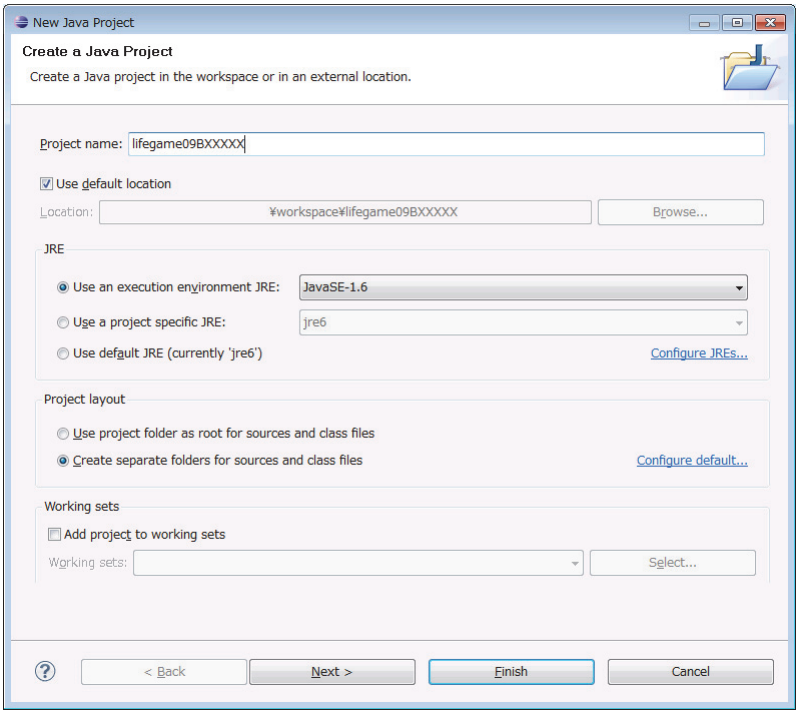

図 11: 新規 Java プロジェクトの情報を入力するダイアログ．本文に記載しているように必要項目を設定したら「終了 (Finish)」を押す.「次へ (Next)」より先の設定は変更しない．

する．

ワークベンチ (Workbench) というのは Eclipse のウィンドウのことである．ワークベンチは複数の「ビュー」に仕切られており，それぞれのビューに異なる情報を表示する．図 10 のスクリーンショットでは，左側に「パッケージ・エクスプローラ (Package Explorer)」「階層 (Type Hierarchy)」ビュー，右側に「アウトライン (Outline)」ビューが表示されており，右下の領域には「問題 (Problems)」「Javadoc」「宣言 (Declaration)」の 3 つのビューが表示されている．

一方，**ワークスペース (Workspace)** は Eclipse のデータを保存するディレクトリのことである．ソースコードやそのコンパイル結果，ウィンドウのレイアウト，オプション設定などがすべて保存され，Eclipse を起動したとき，前回の終了時の状態から作業を再開することができる．なお，Eclipse とは別の方法で直接ソースコードを編集した場合などは，プロジェクトを選択した状態で「ファイル (File)」メニューの「最新の状態に更新 (Refresh)」を実行し，ファイルの最新の状態を Eclipse に読み込ませる必要がある．

1.2 プロジェクトを作る

Eclipse が用意したワークスペースで，ユーザは複数のプログラムを編集することができる．プログラムごとに使用するプログラミング言語やコンパイラ，コンパイルすべきファイルなどが異なるため，Eclipse においては,「プロジェクト」という単位でこのような情報を管理する．巨大なシステムは複数のプロジェクトから構成されることもあるが，本演習ではこれから作る 1 つのプロジェクトを最後まで使用することになるだろう．

作業用のプロジェクトを作成せよ．プロジェクト名は，「`lifegame`」と学籍番号を連結したものとする．たとえば，番号が `09Bxxxxx` ならばプロジェクト名は `lifegame09Bxxxxx` とする．この名前は，最後のプログラム提出時の制約でもある．プロジェクトを作るには,「ファイル (File)」メニューの「新規 (New)」「Java

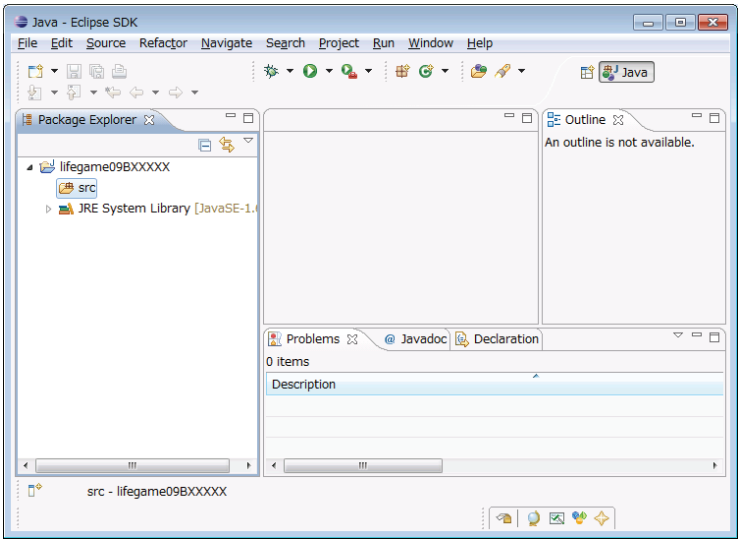

図 12: プロジェクトが追加されたあとのワークベンチの状態の例．プロジェクトはツリー構造になっており，クリックすると中にあるソースコードディレクトリ，ライブラリが表示される．

プロジェクト (Java Project)」を選択する．図 11 に示すダイアログで，新規 Java プロジェクトの情報が要求されるので，以下の項目を指定する．

プロジェクト名 (Project Name) `lifegame` という文字列に学籍番号を連結したものを名前として入力する．

JRE この項目は，コンパイル時および実行時に使用する Java の言語仕様のバージョンを選ぶ．ここでは「Use an execution environment JRE」にある「javaSE-1.6」（あるいは JDK6，JRE6, j2SE-1.6）を選択する [7]．Java は言語仕様や標準ライブラリが順番に拡張されてきているので，古いバージョンを選択すると，本資料のサンプルコードがエラーを起こす場合がある．自宅での学習環境などで新しいバージョンを選んでも本資料のサンプルコードは動作するが，新しいバージョンで導入された記法を使用してプログラムを書くと演習室の環境ではコンパイルできなくなる．

これらを選択したら「終了 (Finish)」を押してプロジェクトの作成を完了する．ワークベンチにプロジェクトが表示されるので，中身を確認すること．プロジェクトはツリー構造になっており，図 12 のようにプロジェクトの中にソースコードディレクトリ，ライブラリが表示される．プロジェクトの名前を間違えてしまった場合は,「ファイル (File)」メニューから「名前変更 (Rename)」を選んで修正すること．

この作業が完了した時点で，Eclipse 起動時に選択したワークスペースのディレクトリ（通常であれば `workspace`）の下に指定したプロジェクト名（たとえば `lifegame09Bxxxxx`）のディレクトリが作成され，プロジェクト情報ファイル（`.project`）などが配置される．多くのファイルはテキスト形式で内容を閲覧できるので，Eclipse の挙動について詳しく知りたい場合は，これらのファイルの内容を確認してみると良い．

1.3 文字コードを設定する

Eclispe の各プロジェクトでは，ソースコードに使用する文字コードを設定できる．この設定を行わないと，OS ごとの文字コードを使用してソースファイルを読み書きしようとするので，自宅などとソースコードを行き来させた場合に文字化けが発生することになる．

[7] 元々の名前が Java 2 Software Development Kit Standard Edition 1.6，別名として JDK6 など複数の呼び名があるため，環境によって表示が異なる．演習室でもソフトウェアのバージョンアップごとに表記が変わっているので，もしかしたらこの通りの名前ではないかもしれない．

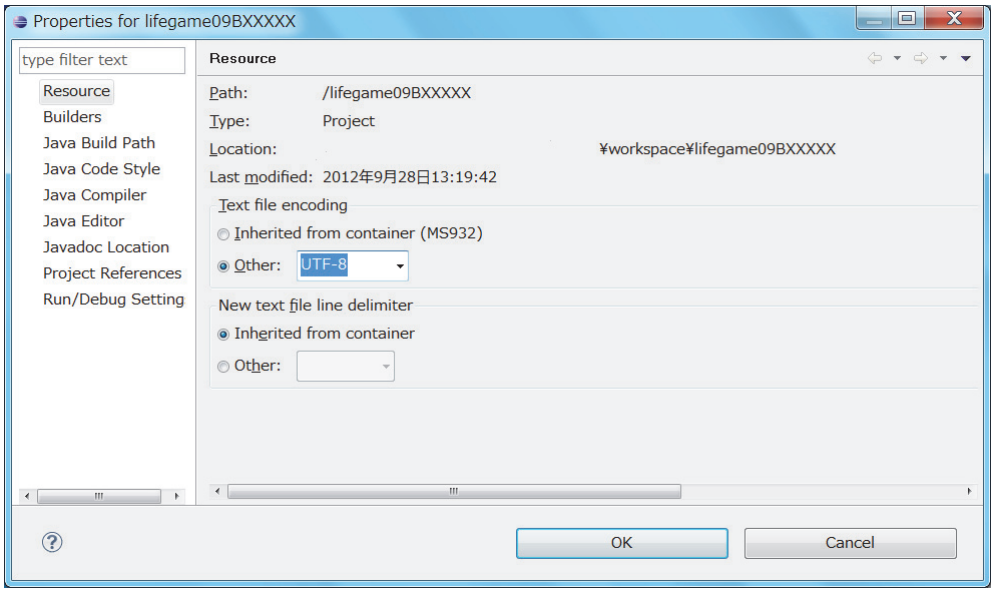

図 13: プロジェクトのプロパティダイアログ．ファイルに使う文字コードをあらかじめ指定しておく．

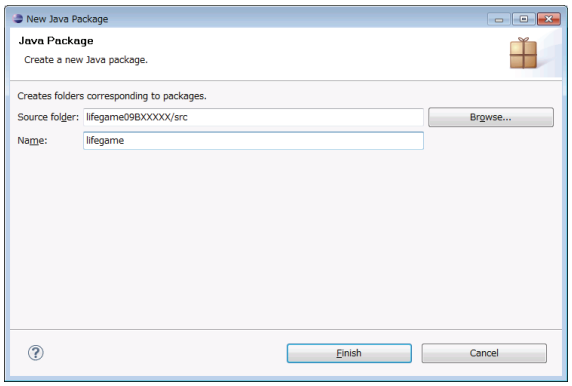

図 14: ファイル (File) メニューの「新規 (New)」「パッケージ (Package)」から，パッケージを追加する．プログラム名を反映して lifegame としておくこと．

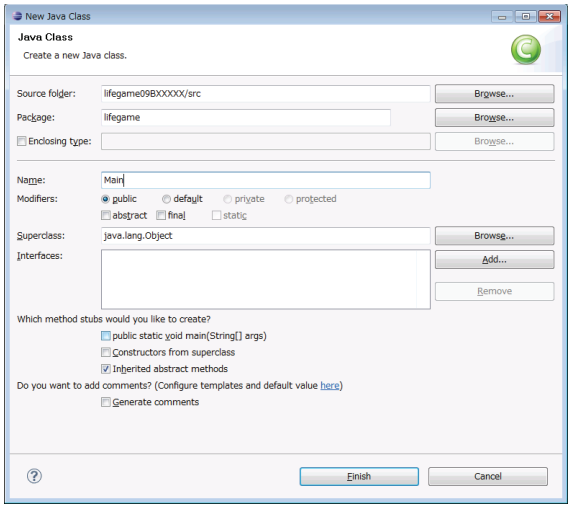

図 15: ファイル (File) メニューの「新規 (New)」「クラス (Class)」から，Main クラスを作成する．

プロジェクトを右クリックして表示されるコンテキストメニューから「プロパティ (Properties)」を選び，表示されるダイアログで文字コードの設定を行うこと．図 13 のようなダイアログが表示されるので，「テキストファイルエンコーディング (Text File Encoding)」を「コンテナから継承 (Inherited from Container)」から「その他 (Other)」に変更し，プロジェクトで使う文字コードを固定する．どの文字コードを使うかについて特に好みがなければ，UTF-8 を指定せよ．

1.4 ベースとなる Java ファイルを作る

プログラム作成の準備として，パッケージ `lifegame` を作成せよ．Java を使ったプログラミングでは，プログラムを 1 ファイルにまとめるのではなく，比較的小さな「クラス」という単位に対応するファイルに書

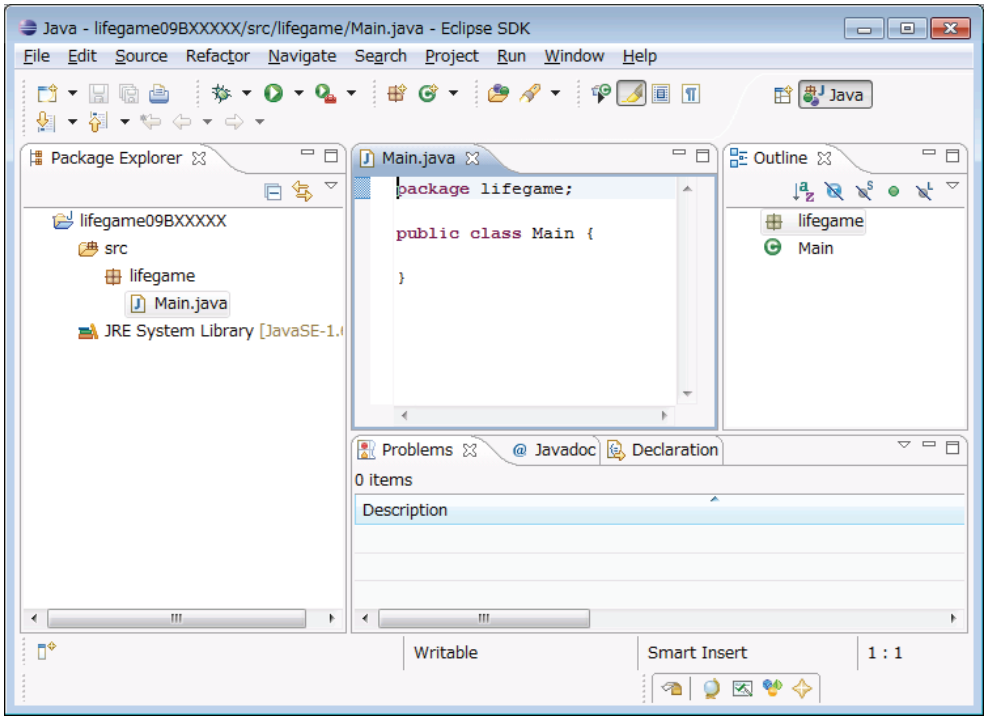

図 16: Main クラスを作成した直後．タブの横にある「×」マークでファイルを閉じることができる．再度開くには 「パッケージ・エクスプローラ (Package Explorer)」の Main.java をダブルクリックする．

き分ける．パッケージとは，Java プログラムを構成するクラスを管理するための単位であり，多数のファイルを整理するためのディレクトリのような存在である．すべてのクラスは「パッケージ `java.lang` のクラス `String`」というようにパッケージ名とクラス名の組によって識別されるため，ライブラリ等に偶然同じ名前のクラスが入っていても問題が起きないように，また，自作プログラムのクラスであることがすぐ分かるように，常に何らかのパッケージを使うことが基本である．本演習では，ライフゲーム用に作成するクラスを入れるパッケージ名を `lifegame` としておく（名前そのものは管理用の情報であり，異なる名前を付けたからといってプログラムの動作に影響が出るわけではない）．機能拡張などによってクラス数が増えない限り，パッケージはこの 1 つだけを使う．

パッケージを作成するには，Eclipse の「ファイル (File)」メニューから「新規 (New)」「パッケージ (Package)」を選ぶ．すると，図 14 のような新規パッケージを作成するダイアログが表示されるので，プロジェクトのソースディレクトリ中に `lifegame` パッケージを作成するよう指定する．この作業が終わると，プロジェクトには `src` ディレクトリ以下にパッケージ名が表示されるようになる．

パッケージ `lifegame` に，プログラムの実行開始点となるクラス `Main` を作成せよ．Java のプログラムは，プログラムのエントリポイントとして指定したクラスに定義された `main` の先頭から実行が開始される [8]．本演習では，レポート提出時の動作確認の都合上， `main` を持たせるクラスを `Main` と規定している．

クラスを作成するには，メニュー「ファイル (File)」「新規 (New)」「クラス (Class)」を選び，図 15 のようなダイアログを開く．一見すると項目数は多いが，実際に設定する必要があるのは以下の 2 項目だけである．これらを指定してから「終了 (Finish)」を選ぶとクラスが作成される．

パッケージ (Package) 先ほど作成した `lifegame` を記入する．

名前 (Name) `Main` と入力する．

[8] 実行時に `main` を持つクラスを指定するので，`main` を持つクラスが複数存在していてもよい．

クラスが作成されると，図 16 のようにクラスに対応するファイル（ここでは Main.java）が作成され，エディタでそのファイルが開かれた状態になる．タブの横にある「×」マークをクリックするとファイルを閉じることができ，また，再度開くには 「パッケージ・エクスプローラ」の Main.java をダブルクリックすればよい．

この時点でファイルを開くと（正しくパッケージ名やクラス名を指定していれば），次のような内容が書かれているはずである．

```
package lifegame;

public class Main {

}
```

ダイアログには細かい設定項目があるが，ここまでの一連の操作で Eclipse が行ったことは，プロジェクトの src ディレクトリ以下に lifegame というパッケージ名に対応するディレクトリを作成し，その中に上記のような内容を持った Main.java ファイルを作っただけである．Java プログラミングにおいては，1 行目の package lifegame; というパッケージ宣言がディレクトリ名と一致していなくてはならず，また，ファイル名とクラス名が一致していなければならないというルールがあるが，ルールに従ったディレクトリやファイルを容易に作れるというのが，Eclipse を用いる利点の 1 つである．

Main クラスの内容は，{ } （ブレース）で括られた範囲である．現状ではまだ何もない状態であるが，この中に Main クラスの内容として必要な内容を書き加えていく．

1.5 Hello, world! プログラムを記述する

Main.java ファイルのクラス宣言に，次のようにmainメソッドを定義せよ．package 宣言と class Main の宣言はファイル作成時に自動的に作られているはずなので，その中に 3 行を追記すればよい．

```
package lifegame;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }

}
```

コードを書き加えてから，ファイルをメニュー「ファイル (File)」「保管 (Save)」（ショートカット CTRL+S）によって上書き保存すると，コンパイルが自動的に実行される．ファイルを保存した時点でファイルに赤線が引かれておらず,「問題 (Problems)」ビューにエラーが出ていなければ，プログラムは実行可能な状態である．

ここで定義した main は，メソッド (Method) と呼ばれる命令の記述単位である．C 言語における main 関数とよく似ているが，外側に class Main { ... }という宣言があるため，このメソッドが Main クラスに所属することを表現している．public static という 2 つのキーワードは，プログラムのいつでも，どこからでもこの処理が呼び出せることを示している．これら 2 つのキーワードより後ろは C 言語での関数の宣言と同じく，戻り値が void 型である（つまり戻り値がない）こと，名前が main であること，引数が String[] 型（文字列の配列）であること，引数の変数名が args であることを示している．

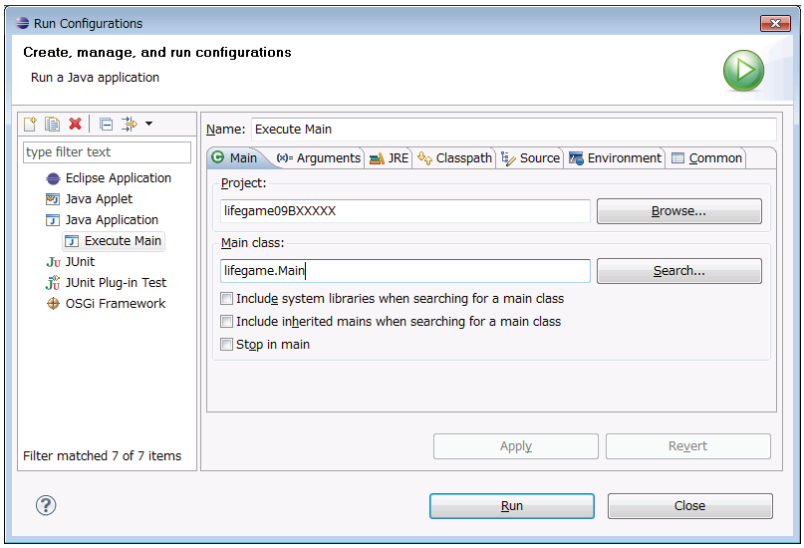

図 17: メニュー「Run」「Run Configurations...」で表示される実行構成ダイアログ．メインクラスの設定欄に `lifegame.Main` と設定して「実行 (Run)」ボタンを押すと，プログラムの実行が開始される．

メソッド名や引数の後ろに続く “{” から対応する “}” までがメソッドの内容であり，C 言語の関数と同様に命令の列を記述する領域である．ここでは，1 行だけを記述した．ピリオド（“.”）でつながれたものは，C 言語の構造体（`struct`）の要素を表現する場合と同様の解釈である．つまり，`a.b` は「`a` の要素の `b`」と解釈する．上記プログラムにおいて `System.out.println(...);` というのは，`System` という名前で参照できる要素（Java の言語標準で定義されている `System` クラス）の `out` という要素（プロセスの標準出力を意味する）の `println` という要素（メソッド）を呼び出す文である．続く括弧は C 言語での関数呼び出しと同様に引数を意味しており，標準出力の `println` メソッドに引数 `"Hello, world!"`という文字列を渡すことで，当該文字列を出力させることを表現している．C 言語の関数呼び出しとの違いは，呼び出し対象（処理の実行を担当する）オブジェクトをピリオドの前で指定していることである．

1.6 プログラムを実行する

作成した `main`メソッドを実行せよ． メニュー「実行 (Run)」から「実行構成 (Run Configurations...)」を選択すると，Java アプリケーションなどを実行（Launch）するための構成，すなわちコマンドラインオプションや周辺環境の設定情報を作成するためのダイアログが表示される．このダイアログでは，左側にリストアップされている項目から「Java アプリケーション (Java Application)」を選び，リスト上側のボタンの並びの左端にある「新規の起動構成 (New Launch Configuration)」ボタンを選ぶと，図 17 のように Java アプリケーションの実行のための設定が作成される．このダイアログには多数の項目が表示されるが，指定する必要があるのは，どのプロジェクトのどのクラスに記述された `main` メソッドからプログラムを実行するか，という点だけである．今回の実行の開始点は，`lifegame` パッケージの `Main` クラスに書かれているので，「メイン・クラス (Main Class)」の欄に `lifegame.Main` を指定する [9]．確認できたら，「実行 (Run)」ボタンを押し，プログラムの実行を開始する．

実行を開始すると，図 18 のように「コンソール (Console)」ビューが開き，文字列が表示される．実行時エラーが発生した場合は，エラーメッセージの 1 行目に直接の原因が書いてあるので，それを確認すること．

[9] パッケージ名の後ろに “.” とクラス名を続けて記述することでクラスを一意に指定する「完全限定名」と呼ばれる記法である．今後も登場するので覚えておくこと．

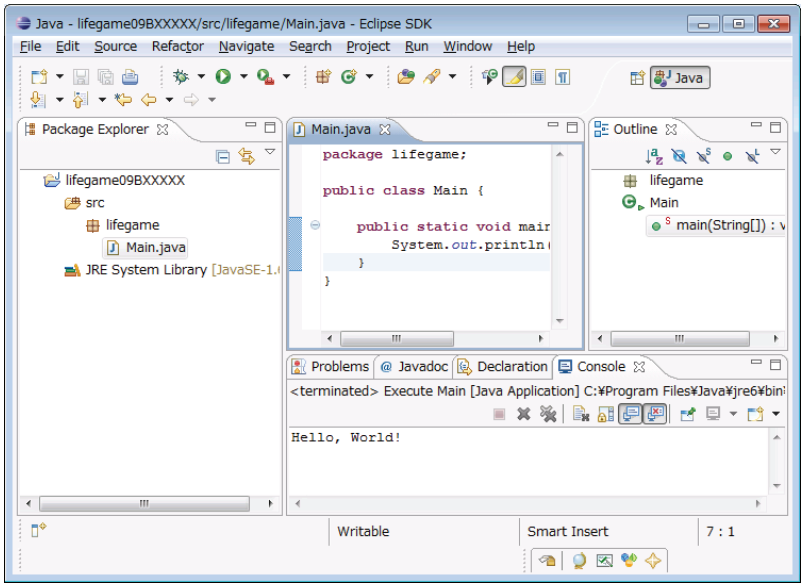

図 18: Hello, world!

Main クラスにコンパイルエラーが出ていなければ，指定したパッケージ名，クラス名が誤っている可能性が高い．

1.7 ここまでのまとめ

このステップでは，Eclipse でプロジェクト，パッケージ，クラスを作成し，プログラムを編集，保存，実行する手順を確認した．本資料では，これ以降，新しいクラスを作成する操作は既知のものとして扱う．

Eclipse について覚えておくと良い特徴は，以下の通りである．

- 設定ファイルやソースコードはすべて workspace ディレクトリに保存される．Eclipse の設定がおかしくなった場合は，ソースコードだけ退避して，新しい workspace を作成しなおせば良い．
- 入力するプロジェクト名やパッケージ名，クラス名を変更することで，自分の作業に必要な様々なファイルを自由に作ることができる．
- クラスを作成するダイアログでは，項目の選択を変更すると，作成されるファイル（クラス定義）の内容が変化する．作成した後に自由に中身を編集することができるし，ファイルの削除もできるので，設定を変えてみて，どのようなファイルが出来上がるかを確かめてみると良い．
- ソースファイルを保存すると，自動でコンパイルが行われる．コンパイルエラーは「問題 (Problems)」タブに表示される．
- プログラムの実行も Eclipse から指示することができる．プログラムの実行の設定を行うダイアログでは設定項目の数が多いが，Java プログラムを実行する際にコマンドラインオプションから指定できる項目と内容は同じである．プログラムの実行の開始点となる main メソッドを持つクラスの指定と，必要な場合の引数の指定以外は，標準の設定で問題なく実行できるはずである．

時間に余裕があれば，メニューに書かれているショートカットキーや，ツールバーのボタンの意味を調べてみると良い．これらをうまく活用すると，素早く作業を進めることができる．

2　ライフゲームの盤面となる BoardModel クラスを定義する

このステップでは，ライフゲームを計算機上で実行する第一歩として，Java におけるプログラムの基本構成要素である「クラス」の作成を体験する．

2.1　BoardModel クラスを宣言する

lifegame パッケージに BoardModel クラスを新しく作成せよ．Java では 1 ファイルに 1 クラスだけを記述することが基本となるので，Main と同様の手順で新しく作成しておくこと．新しく作成したばかりの BoardModel クラスは，ブレース（{ }）で囲まれた範囲に何も記述されていないので，この時点ではまだデータも処理も持っていないことになる．

2.2　BoardModel に盤面の大きさを持たせる

ライフゲームを実行する際の盤面のサイズをたとえば 10 × 10 にするか，20 × 15 にするか，というのはいつ決めても問題ないが，あとからでも変更可能なように変数で表現してプログラムを書くことを考える．盤面の水平方向の列数を cols，垂直方向の行数を rows と呼ぶことにして,BoardModelクラスに，これらを int 型のフィールドとして定義せよ．フィールドはクラスに所属しなくてはならないので，BoardModel クラスの宣言の内部（ { } で囲まれた範囲）に以下のように記述を行う．

```
private int cols;
private int rows;
```

この定義が加わっただけでは，プログラムの動きに変化はない．

2.3　盤面の大きさを設定するためのコンストラクタと，参照のためのメソッドを定義する

定義した cols, rows フィールドは，いわゆる変数の一種なので，値を代入する処理，参照する処理の両方を定義しなくては意味がない．そこで，まず代入処理として，BoardModelクラスの内部に，以下のようにコンストラクタを定義せよ．

```
public BoardModel(int c, int r) {
    cols = c;
    rows = r;
}
```

コンストラクタは，C 言語における関数の一種であるが,「オブジェクトが作成されたとき」に呼び出される処理である．コンストラクタの宣言は，可視性の指定，コンストラクタであることを示すクラス名，引数の宣言と本体からなる．可視性としては，プログラムのどこからでも使えることを意味する public を指定した．クラス名は，当然ながら BoardModel である．引数に盤面の列数，行数を指定した整数を取り，コンストラクタ中でそれをフィールドに代入する．

続いて，フィールドを外部から読み出すための処理として，BoardModelクラスの内部に，以下のように 2 つのメソッドを定義せよ．

```
public int getCols() {
    return cols;
}
```

```
    public int getRows() {
        return rows;
    }
```

メソッドは，コンストラクタと同様に C 言語における関数の一種であり，作成されたオブジェクトに対して自由なタイミングで呼び出すことができる．可視性が加わったことを除けば，戻り値の型，メソッド名，引数のリスト（ここでは引数なし），メソッド定義本体かなるという点で C 言語における関数と同様の表現である．これらのメソッドを実行すると，それぞれ cols, rows フィールドの値を return 文によって返却する．

これらの定義を加えても，やはりプログラムの動きに変化はない．C 言語において，新しく関数をいくら宣言しても，それを呼び出さない限り効果がないのと同じである．

2.4 Main 側で盤面オブジェクトを作成する

盤面オブジェクトの存在を確認するために,mainメソッドの内容を以下のように変更し，実行せよ．

```
    public static void main(String[] args) {
        BoardModel model1 = new BoardModel(12, 10);
        System.out.print(System.identityHashCode(model1));
        System.out.print(",Cols=" + model1.getCols());
        System.out.println(",Rows=" + model1.getRows());
        BoardModel model2 = new BoardModel(20, 15);
        System.out.print(System.identityHashCode(model2));
        System.out.print(",Cols=" + model2.getCols());
        System.out.println(",Rows=" + model2.getRows());
    }
```

プログラムを実行すると，次のように出力が得られるはずである（xxxxxxxx, yyyyyyyy のところには何らかの数値が入る）．

```
xxxxxxxx,Cols=12,Rows=10
yyyyyyyy,Cols=20,Rows=15
```

new はオブジェクトの新規作成を指示する予約語である．

```
        BoardModel model1 = new BoardModel(12, 10);
```

という行は，C 言語であれば malloc 関数に相当する処理としてメモリ領域の確保を行い，それからコンストラクタに引数 12, 10 を渡して実行する．コンストラクタの中の代入文は，確保されたメモリの cols と rows にその値を代入する．そして，作成されたオブジェクトへの参照が変数 model1 に代入される．

一時的な警告

クラスの外からアクセスできない private なフィールドを新しく追加したときには，まだどこからもその値を参照していないので,「The field _____ is never read locally」あるいは「The value of the field _____ is not used」といった警告が表示されることがある．コードを書き進めていくと警告は消えるはずなので，出てきた時点では気にする必要はない．もしフィールドの値を参照しているつもりなのにこの警告が消えていない場合は，そのフィールドの値が正しく参照できていない（たとえば同名のローカル変数を誤って参照している）可能性がある．

ここで重要なのは，BoardModel というクラス定義は 1 つであり，cols, rows という変数宣言もそれぞれ 1 つずつであるが，実行時には BoardModel オブジェクトを複数作ることができ，cols, rows を記憶する領域はオブジェクトごとに準備されている，という事実である．

「オブジェクトへの参照」は，C 言語的な表現でいえば，確保されたオブジェクトへのポインタであると思っておくとよい．つまり，

```
BoardModel model3 = model1;
```

という文を実行した場合は，変数 model3 もまた同一の BoardModel オブジェクトを参照する（コピーが作られるわけではない）．ポインタの値を直接見ることはできないが，System.identityHashCode というメソッドを通すとオブジェクトへの参照を数値に変換して確認することができる．System.identityHashCode で出力される数値は，オブジェクトを作る個数や順序が変わるとすぐに変化するので，オブジェクトがあくまでも「実行時に」作られるものであることを意識しやすい．

「オブジェクトへの参照」を使って，メソッド呼び出しを行うことができる．これは C 言語における関数呼び出しに似ているが，model1.getCols() というように，オブジェクトを左側に書き，「model1 が参照しているオブジェクトの」getCols メソッドを呼び出すという形に解釈する．model1 の getCols メソッドの実行では model1 を作ったときに代入された cols フィールドの値（12）が，model2 の getCols メソッドの実行では model2 を作ったときに代入された cols フィールドの値（20）が，それぞれ返却され，文字列として出力される．

new は，1 回実行するごとに，オブジェクトを 1 つ作成する．1 つの文で複数のオブジェクトを作る例としては，以下のようなコードが挙げられる．

```
public static void main(String[] args) {
    for (int i=0; i<10; i++) {
        BoardModel model1 = new BoardModel(10+i, 10);
        System.out.print(System.identityHashCode(model1));
        System.out.print(",Cols=" + model1.getCols());
        System.out.println(",Rows=" + model1.getRows());
    }
}
```

「オブジェクト」の概念に対する理解が Java プログラミングの核となるので，ここで様々な操作を試してみると良い．

2.5 ここまでのまとめ

このステップで作成した「オブジェクト」が，プログラムの基本構成要素である．ここで以下の概念を理解することが望ましい．

- オブジェクトは「フィールド」を持つことができる．このフィールドは，オブジェクトを 1 つ作成するごとに，そのオブジェクト専用の領域として確保される．
- オブジェクトは「コンストラクタ」を持つことができる．コンストラクタは，new によってオブジェクトを作成するときに呼び出される関数の一種であり，フィールドに初期値を代入する処理などを記述する．
- オブジェクトは「メソッド」を持つことができる．メソッドはオブジェクトに所属する関数の一種であり，そのオブジェクトのフィールドを読み書きすることができる．オブジェクトを一度作成したあとは，そのオブジェクトへの参照を使って，メソッドを何度でも呼び出せる．

- 「クラス」の中に，そのオブジェクトが持つフィールド，コンストラクタ，メソッド一式を定義する．クラスが定義されると，new によってそのオブジェクトを作成できる．

3 ライフゲームの盤面としての機能を BoardModel クラスに付け加える

盤面には「サイズ」だけでなく，そのセルの状態も持たせなければならない．そこで，盤面のデータを表現するデータ構造を 2 次元配列として定義するとともに，配列操作の練習として，配列の内容を文字列として標準出力に書き出すプログラムを作成する．

3.1 盤面を表現する配列を格納するフィールドを宣言する

ライフゲームにおいて，セルは生きているか死んでいるかの 2 値を取るので，Java の boolean 型で表現することができる（使用例は FAQ Q13 参照）．BoardModelクラスに，boolean の二次元配列型のフィールド cells を定義せよ．BoardModel クラスの宣言の内部（ { } で囲まれた範囲）に以下のように記述を行う．

```
private boolean[][] cells;
```

boolean[][] と cells は，それぞれ変数の型と名前に対応し，セミコロンで宣言を完了している（C 言語と同様に省略できない）．boolean[][] は，boolean 型の配列を意味する boolean[] が配列になっていることを意味する．2 次元配列は，盤面の作成と同時に作ることができる．たとえば次のように，コンストラクタに 1 行追記するというのが 1 つの方法である．

```
public BoardModel(int c, int r) {
    cols = c;
    rows = r;
    cells = new boolean[rows][cols];
}
```

型名と配列のサイズを指定して作成された配列への参照をフィールド cells に代入している．

盤面は縦横にセルが並んでいる状態となっているので，上記の配列の作り方だと，整数の組 x，y で表現される位置のセルを配列の cells[y][x] に対応付ける記述となっている．X 軸，Y 軸は，ウィンドウ上での座標軸に合わせて，右方向を X 軸，下方向を Y 軸とした表記を用いる．つまり，一番左上のセルが cells[0][0]，その右隣が cells[0][1]，というように対応する．

可視性によるデータ保護

フィールド cols, rows は private として宣言したので，BoardModel クラスの外からは参照できない．たとえば，Main クラスに所属する main メソッドの中では，

```
BoardModel model1 = new BoardModel(12, 10);
System.out.print(",Cols=" + model1.cols);
```

という記述はコンパイルエラーとなる．時間があれば確認してみること．
このようなフィールド参照がコンパイルエラーになるということは，BoardModel を作ったときに渡した盤面の大きさは，このオブジェクトの外側からは勝手に変更できないということである．そのため，BoardModel の中でメソッドの内容を読み書きするとき，知らないところで盤面の大きさが勝手に変わることはないと確信をもって作業を行うことができる．

3.2 配列の内容を書き出すメソッドを宣言する

BoardModelクラスの内部に，以下のようにメソッドを定義せよ．

```
public void printForDebug() {
    System.out.println(cells[0][0]);
}
```

このメソッドに，盤面の配列の内容を書き出させる処理を記述する予定である．現時点では，動作確認用に配列の左上の要素を出力する 1 行だけを記述して，先に呼び出し部分を作成する．

3.3 main からオブジェクトの作成を行い，メソッドを呼び出す処理を記述する

Mainクラスのmainメソッドを以下のように書き換えよ．

```
public static void main(String[] args) {
    BoardModel model = new BoardModel(10, 10);
    model.printForDebug();
}
```

ここで記述した 2 つの文は，まず，BoardModel クラスの定義からオブジェクトを 1 つ作成し，model という変数に代入する．次に，そのオブジェクトに対して，printForDebug メソッドを呼び出す．

このプログラムを実行すると，画面には false という文字列が出力されるはずである．改めて命令の実行順序を確認すると，次のようになる．

1. Main クラスの main メソッドの実行が開始される．
2. new によって指定された BoardModel オブジェクト用のメモリ領域が確保される．
3. BoardModel のコンストラクタが呼び出される．
4. BoardModel オブジェクトの cols, rows 各フィールドに，new の引数である 10 が代入される．
5. コンストラクタに記述された new 命令によって 2 次元配列が作成され，その配列への参照が cells フィールドに代入される．
6. コンストラクタの実行が完了したので，main に制御が戻る．
7. コンストラクタの実行が完了して使用の準備が整った BoardModel オブジェクトへの参照が，ローカル変数 model に代入される．
8. 次の行に進み，model に格納された参照を通じて，作成された BoardModel オブジェクトのメソッド printForDebug を呼び出す．
9. printForDebug の中で cells を参照し，得られた false という値を標準出力に書き出す．
10. main に制御が戻る．
11. main の最後の文まで実行が完了したので，プログラムが終了する．

3.4 配列の中身を書き出す処理を定義する

BoardModel クラスの printForDebug メソッドの中身として，cells に格納されたセルの情報を標準出力に書き出す処理を記述せよ．配列の要素として true が入っていれば生きているセルとしてアスタリスク（*）を，false が入っていれば死んでいるセルとしてピリオド（.）を書き出すこと．また，printForDebug を連続して実行できるように，盤面の終端には空行を 1 行出力せよ．たとえば 10 × 10 の空の盤面を対象として printForDebug を実行した場合，改行記号を「↓」で表記すると，次のような出力を行う．

```
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
.......... ↓
↓
```

3.2 節でも記述しているように，配列の要素にアクセスする方法は C 言語とほぼ同様である．ループには for 文，while 文が使用できるし，条件分岐には if 文が使用できる．配列の各次元の長さは（もしここまで資料どおりに進んでいれば）cols, rows フィールドに格納されている．

文字列の出力には，これまでにも何度か登場している標準出力オブジェクト System.out のメソッドが使用できる．主な機能を以下に示すが，詳細は java.io.PrintStream クラスのドキュメントを参照すること．

- System.out.print("c"); のように引数を持つ print メソッドの呼び出しは，改行なしで文字列を出力する．
- System.out.println("c"); のように引数を持つ println メソッドの呼び出しは，引数で指定された文字列を書き出した後，改行を出力する．
- System.out.println(); のように引数を持たない println メソッドの呼び出しは，改行だけを出力する．
- print() と println() は，数値など，他の型のデータに対しても定義されている（オーバーロード）．たとえば print(1) のように数値を出力するためにも使用できる．

なお，ダブルクォートで囲んだ文字列はすべて String 型である．String s = "*"; のように変数に代入しておき，print メソッドの引数にすることができる．また，文字列は + 演算子によって連結することもできる．

メソッドの記述が完成したら，必ず実行して，配列として確保した数だけピリオドが出力されることを確認すること．ArrayIndexOutOfBoundsException というメッセージが出力された場合は，配列の範囲外にアクセスしている．配列の添え字の計算に何らかの間違いがあるので，表示される行番号を手がかりに確認すること（エラーメッセージの意味は FAQ Q10 参照）．

3.5 盤面の状態を変更する操作を定義する

空の盤面を表示することができるようになったら，次は生きているセルを配置する処理を記述する．最終的には，マウス操作で盤面の状態を変更しなくてはならないので，BoardModel 外部から引数として変

更の指示を受け取れるようにする必要がある．そこで，盤面の指定されたセルの状態を変更するメソッド changeCellState を BoardModel に定義せよ．引数は，変更したいセルを選ぶ int の組 x，y とする．セルの状態を変更するというのは，セルが生きているなら死んでいる状態に，死んでいるなら生きている状態に変更するということである．定義すべきメソッドの宣言は次のようになる．中身は各自で埋めること．

```
public void changeCellState(int x, int y) {
    // (x, y) で指定されたセルの状態を変更する.
}
```

盤面の操作を確認できるように,mainメソッドの内容を以下のように拡張せよ.

```
public static void main(String[] args) {
    BoardModel model = new BoardModel(10, 10);
    model.printForDebug();
    model.changeCellState(1, 1);
    model.printForDebug();
    model.changeCellState(2, 2);
    model.printForDebug();
    model.changeCellState(0, 3);
    model.printForDebug();
    model.changeCellState(1, 3);
    model.printForDebug();
    model.changeCellState(2, 3);
    model.printForDebug();
    model.changeCellState(4, 4);
    model.printForDebug();
    model.changeCellState(4, 4);
    model.printForDebug();
}
```

changeCellState の内容を正しく実装していれば，このプログラムを実行すると，盤面が変更していく様子が標準出力に書き出されるはずである．最初の 2 回と最後の 2 回で出力される盤面の状態を，参考として次に示す（実際にはこのような盤面の変化が縦方向に連なって出力されることになる）.

```
..........      ..........              ..........      ..........
.*........      .*........              .*........      .*........
..........      ..*.......              ..*.......      ..*.......
..........      ..........              ***.......      ***.......
..........  ->  ..........  -> (...) ->  ....*.....  ->  ..........
..........      ..........              ..........      ..........
..........      ..........              ..........      ..........
..........      ..........              ..........      ..........
..........      ..........              ..........      ..........
..........      ..........              ..........      ..........
```

3.6 ここまでのまとめ

この時点で，盤面を main メソッドから操作し，その盤面の状態を出力できるプログラムが出来上がったことになる．配列がオブジェクトごとに作られることを確認したい人は，前章と同様に，独立した BoardModel オブジェクトをもう 1 つ作って，それに対しても changeCellState，printForDebug を呼び出してみるとよい．

4 盤面の状態が更新されたときに処理を実行する仕組みを作る

現在のプログラムでは，盤面の状態が更新されたとき，逐一 printForDebug() メソッドを呼び出して画面を更新する必要がある．しかし，盤面の状態が更新されたかどうかは，本来，Main 側ではなく，BoardModel 側が知っている情報であり，外部から適切なタイミングで呼び出すには手間がかかる場合もある．このステップでは，Java のインタフェースを導入し，盤面の状態が更新されたときに実行するべき処理を事前に BoardModel に「登録」しておき，必要なときに盤面側から呼び出せるようにする仕組みを構築する．インタフェースという言語の仕組みに従うため，記述するコード自体の自由度はほとんどない．

4.1 インタフェース BoardListener を宣言する

以下に示すインタフェース BoardListener を作成せよ．インタフェースとは，Java において，クラスがある特定のメソッドを所有することを保証するための言語要素である．BoardModel から「通知を受け取りたい」と考えるオブジェクトのことを BoardListener と呼ぶことにして，それらのオブジェクトが所有するべきメソッドを定義する．

インタフェースの作成は，メニューから「インタフェース（Interface）」の作成を選ぶ以外は，クラス作成の手順と同じである（クラスを作成してからキーワード class を interface に置き換えても良い）．BoardListener.java ファイルとして作成すべきコードを以下に示す．

```
package lifegame;

public interface BoardListener {

    public void updated(BoardModel m);

}
```

内部には呼び出す予定のメソッドの名前と引数，戻り値を宣言する．ここでは updated という名前のメソッドとして，引数に更新された BoardModel が入るものとする．メソッドに本体はなく，セミコロンでメソッドの名前だけを記述する．

配列へのアクセスをメソッドで隠ぺいする理由

普通にセルを 1 対 1 に対応付ける，つまり cells[y][x] の値を書き換えるだけであれば，changeCellState のようなメソッドの定義は単にプログラムを長くしているだけのように見える．しかし，changeCellState のように意味のある処理をメソッドに独立させておくことで，引数である x や y が正しい範囲かどうかをチェックしたり，GUI を実装する際に「盤面の状態が変わったら画面の表示を更新する」処理を起動したりといった追加の役割を持たせることができるようになる．

この BoardListener インタフェースの定義は，上記の宣言に一致する updated メソッドを持つクラスのオブジェクトを BoardListener 型として扱えるということを意味する．インタフェースを宣言しただけでは，プログラムの動きに変化はない．また，正しく宣言されていれば，コンパイルエラーも発生しない．

4.2 BoardModel に BoardListener を登録する仕組みを作る

BoardModel クラスに，次のように BoardListener 型のオブジェクトを扱うためのフィールドとメソッドを追加せよ．なお，左端の行番号は解説のために付与したものであり，プログラムの一部ではない．

```
 1:    private ArrayList<BoardListener> listeners;

       public BoardModel(int c, int r) {
           (前のステップで作成した配列の初期化処理はここでは省略)
 2:        listeners = new ArrayList<BoardListener>();
       }

 3:    public void addListener(BoardListener listener) {
 4:        listeners.add(listener);
 5:    }

 6:    private void fireUpdate() {
 7:        for (BoardListener listener: listeners) {
 8:            listener.updated(this);
 9:        }
10:    }
```

上記コードの 1 行目は，BoardListener 型のオブジェクトを格納できる可変長配列のフィールドの宣言である．使用している型 ArrayList の正式名称は java.util.ArrayList である．以下の import 文を BoardModel.java ファイルのパッケージ宣言の直後，クラス宣言よりも前に追加すること．

```
import java.util.ArrayList;
```

2 行目は，BoardListener を格納するためのオブジェクトを作成しフィールドに代入する処理を既存のコンストラクタに追記している．

コードの 3 行目から 5 行目までは，盤面の状態が更新されたときにメソッドを呼び出されたいオブジェクトの登録を受け付ける addListener というメソッドを定義している．BoardListener 型のオブジェクトへの参照を引数として受け取り listeners に追加する．BoardListener が複数登録できるようにしているのは，今後の機能拡張に備えているためである．

コードの 6 行目から 10 行目までが，更新を伝達するためのメソッドである．listeners リストに追加された各オブジェクトに対して，updated メソッドを呼び出している．引数の this は，自分自身を表現する予約語で，更新のあった BoardModel 自体が引数となることを意味している．ここで使っている for 文は，C 言語で用いている記法とは異なる拡張構文である．ループに使用する変数の型と名前を指定し，コロンの後ろに配列やリストを指定することで，その配列やリストの各要素を指定した変数に代入し，ループの中身を実行する．

正しくプログラムを記述できていれば，これらの記述が完了した時点でコンパイルエラーは発生しない．また，定義したメソッドを呼び出す処理はまだ追加していないので，プログラムの動きにも大きな変化はない．

Eclipse の 自動 import 機能

Eclipse では，メソッドの記述途中でクラス名を（たとえば ArrayList と）入力したあと，そのままコード補完を行う（演習室の FreeBSD 環境では ALT+"/"キー，他の多くの環境では CTRL+Space キー）と，指定されているクラス名がライブラリの中でもただ 1 つの場合は，ただちに import 文が追加される．同名のクラスが複数ある場合は，リストから選んで Enter を押すと，そのクラスの import 文が追加される．間違ったクラスを import してしまった場合は，追加された文を直接修正すればよい．
自動的に対応するクラスを探索し import 文を生成してくれるので，import 文を手書きする必要はほとんどない．ソースコードに構文エラーがある場合はコード補完が失敗することもあるので，その場合はコードを確認すること．

4.3 changeCellState メソッドから，通知を行う処理を呼び出すようにする

BoardModelクラスの changeCellState メソッドに，盤面の状態が変わったとき fireUpdate メソッドを呼び出す処理を追記せよ．同一クラスの内部でのメソッド呼び出しなので，メソッド呼び出し対象のオブジェクトは this である（省略することもできる）．前の章のコードと合わせて，以下のようにコードが並ぶことになるはずである．

```
public void changeCellState(int x, int y) {
    // 前の章で記述した，(x, y) で指定されたセルの状態を変更する処理
    // ここで fireUpdate を呼びだす
}
```

4.4 盤面の更新ごとに盤面の状態を書き出すオブジェクトを登録する

盤面の更新通知を受け取る処理を実装する ModelPrinter クラスを定義し，更新通知を受け取ったらprintForDebug を呼び出す処理を記述せよ．ここは単純に次のようなコードとなる．

```
package lifegame;

public class ModelPrinter implements BoardListener {

    public void updated(BoardModel model) {
        model.printForDebug();
    }

}
```

クラス名に続く implements 節によって，このクラスが BoardListener インタフェースを持つことを宣言している．このインタフェースは，クラスが updated メソッドを持つことを要求しているので，クラスを書きかけ始めてから updated メソッドの定義が終わるまではコンパイルエラーになってしまう．一時的なエラーは気にせず，クラスの作成を完了すること．

この時点でも，まだ ModelPrinter オブジェクトを作成・登録する処理を記述していないため，プログラムの動作は変わっていない．

4.5 BoardModel に ModelPrinter を登録する

Mainの記述を修正し,BoardModelに対してaddListenerを呼び出し,ModelPrinterを登録するコードを追加せよ．また，不要になったprintForDebug呼び出しを削除せよ．具体的には，以下のようなコードになる．

```
public static void main(String[] args) {
    BoardModel model = new BoardModel(10, 10);
    model.addListener(new ModelPrinter()); // 新規に追加する行
    model.changeCellState(1, 1);
    model.changeCellState(2, 2);
    model.changeCellState(0, 3);
    model.changeCellState(1, 3);
    model.changeCellState(2, 3);
    model.changeCellState(4, 4);
    model.changeCellState(4, 4);
}
```

プログラムを実行し，動作を確認せよ．ここまでの記述がすべて正しければ，new ModelPrinter() によって作成されたオブジェクトへの参照が addListener によって model に引き渡され，listeners に格納される．続く changeCellState メソッドが内部で fireUpdate メソッドを呼び出すと，fireUpdate が listeners に格納されている ModelPrinter オブジェクトの updated メソッドを呼び出す．結果として，changeCellState メソッドを 1 回呼び出すごとに，printForDebug メソッドが 1 回呼び出されるようになっていれば，上記のように changeCellState を並べるだけで，盤面の状態が変化する様子が出力されるようになる．

4.6 ここまでのまとめ

このステップで，changeCellState を呼び出すごとに printForDebug が呼び出される仕組みが達成できた．単純に changeCellState から直接 printForDebug を呼び出す場合に比べると大きな手間がかかっているが，この方法のポイントは 4.2 節の時点，つまり実行したい処理を記述するよりも前に呼び出し側を記述できることにある．BoardListener は，以降のステップでも「盤面が更新されたとき」実行したい処理を登録するために使用できる．

ModelPrinter クラスが BoardListener インタフェースを実装しているため，ModelPrinter オブジェクトへの参照を BoardListener 型の変数に代入できるようになっている．GUI の作成においては，ボタンやマウスの操作をプログラム側が受け取るために，標準ライブラリが提供するインタフェースを実装したクラスを準備し，GUI クラスに対して登録することになる．ライブラリ内部で行われていることは，この章で作成した BoardListener の管理・登録の仕組みとほぼ同様なので，GUI を作成する段階になったら改めてインタフェースの意義を考えると良いだろう．

ArrayList を使用したことで，BoardModel には複数の BoardListener を登録することができる．たとえば盤面の状態をファイルに書き出す機能を ModelFilePrinter クラスとして実装したとすると，次のように ModelPrinter と併用することができる．

```
public static void main(String[] args) {
    BoardModel model = new BoardModel(10, 10);
    model.addListener(new ModelPrinter());
    model.addListener(new ModelFilePrinter());
    model.changeCellState(1, 1);
```

```
        ...
    }
```

そのほかにも，盤面の状態に変化があるごとに何らかの処理を行いたい場合に，それらの処理を独立して記述し，追加することができる．

5　ライフゲームの進行処理を実現する

現時点で，プログラムは，main メソッドから changeCellState メソッドを呼び出すことで特定の盤面を準備する機能と，標準出力に盤面の状態を逐次出力する機能を持っている．これらの機能をベースにして，ライフゲームの進行処理，すなわちゲームのルールに従って盤面の状態を更新する処理を実装できる．

このステップは，前のステップとは対照的に，自分で考えてプログラムを組むステップである．

5.1　next メソッドを定義する

BoardModelクラスに，1 回呼び出すごとに盤面の状態をライフゲームのルールに沿って 1 世代更新してから fireUpdate を呼び出す処理を実行するメソッドを next という名前で定義せよ．記述におけるポイントは次の通りである．

- 「次の世代」の状態は，現在の世代の状態によってのみ決まる．つまり，「次の世代」の状態の計算が完了するまでは，現在の世代のデータを捨てたり上書きしたりしてはいけない．
- 新しい配列などが必要な場合は new によって作成することができる．
- Java では，使用されていないメモリ領域は自動的に解放されるので，C 言語での free に当たる明示的な解放は不要である．
- next の実装にあたって，サブルーチンのようなものが必要な場合は，BoardModel クラス中に private 指定のメソッドを自分で定義すればよい．

動作確認を行うために，main メソッドの末尾に，たとえば次のように next を呼び出す処理を追加せよ．

```
for (int i=0; i<12; ++i) {
    model.next();
}
```

このメソッド呼び出しで，changeCellState で用意した盤面の状態から，12 世代，盤面を更新することになる．4.5 節の main メソッドで用意した盤面の場合，プログラムが正しく動いていれば，12 世代後は次のような盤面になる．

```
..........
..........
..........
..........
....*.....
.....*....
...***....
..........
..........
..........
```

5.2 ここまでのまとめ

このステップで，ようやくライフゲームを（盤面を main にメソッド呼び出しの形で記述する必要はあるが）遊べるようになった．時間に余裕があれば，世代の更新回数を増やして動きがどうなるか，また，盤面の端に配置したセルが予想通りに動くかなど，様々な状況を試してみると良い．

6 ライフゲームの巻き戻し処理を実現する

このステップでは，盤面を 1 操作前の状態に巻き戻す機能を実現する．世代を巻き戻すには，過去の盤面の状態を保管しておき，それを必要に応じて取り出すことになる．

6.1 undo メソッドを定義する

BoardModelクラスに，1 回呼び出すごとに盤面を巻き戻すundoメソッドを定義せよ．注意すべきポイントは次の通りである．

- 過去の盤面のデータを保存するためのデータ構造として， java.util パッケージのクラスが使用できる．
- 現時点での changeCellState は，一般的な実装であれば盤面の配列データを直接編集しているはずなので，変更前の盤面情報のコピーを残すように変更しなければならない．二次元配列は「一次元配列への参照を要素とする配列」なので，複製には注意が必要である（FAQ Q15 参照）．
- プログラムへの要求として，データを保管する世代数は限定してある．仕様をよく確認し，無限にデータを保管し続けないように注意すること．
- 巻き戻しを行ったときも， fireUpdate の実行が必要である．

動作確認は，例によって main メソッドに処理を記述すればよい．たとえば以下のようなコードで，盤面が初期状態に戻るかどうかを確認してみると良い．

```
for (int i=0; i<19; ++i) {
    model.undo();
}
```

また，最大巻き戻し回数を確認する場合は，next の呼び出し回数と undo の呼び出し回数を増やせば良い．

6.2 isUndoable メソッドを定義する

BoardModel クラスに，盤面が巻き戻せるかどうかを boolean で返す isUndoable メソッドを定義せよ．このメソッドを定義できれば，main メソッドの巻き戻し処理を，以下のように簡略化することができる．

```
while (model.isUndoable()) {
    model.undo();
}
```

このメソッドを作っておくと，GUI 上で Undo ボタンの有効化，無効化を切り替える際にも使用することができる．

6.3 ここまでのまとめ

セルの状態の変更，ライフゲームの進行と巻き戻しが実現できたので，ライフゲームを支えるデータ構造としては十分である．あとはユーザからの入力を受け付け，BoardModel を操作する処理を記述していくことになる．

7 ウィンドウを作りライフゲームの盤面を表示する

このステップでは，GUI 実現の第一歩として，起動すると main メソッドで作成している盤面を表示するプログラムを作成する．ウィンドウの作成方法は API に沿った制限の強いものであるが，盤面を実際に描画する方法については各自で考える必要がある．

7.1 ウィンドウを作る

ライフゲームのウィンドウを表示するには，まずウィンドウを作成しなくてはならない．Main クラスに定義した mainメソッドから next と undo に関する処理を取り除き，以下のように内容を変更せよ．なお，左端の行番号は解説のために付与したものであり，プログラムの一部ではない．

```
public class Main implements Runnable {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Main());
    }

    public void run() {
        // BoardModel の作成と changeCellState の呼び出しを行う処理をここで実行.
        // next と undo の呼び出しを取り除き,「グライダー」が設置された状態としておく.

        // ウィンドウを作成する
 1:     JFrame frame = new JFrame();
 2:     frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // ウィンドウ内部を占有する「ベース」パネルを作成する
 3:     JPanel base = new JPanel();
 4:     frame.setContentPane(base);
 5:     base.setPreferredSize(new Dimension(400, 300)); // 希望サイズの指定
 6:     frame.setMinimumSize(new Dimension(300, 200));  // 最小サイズの指定

 7:     frame.pack();            // ウィンドウに乗せたものの配置を確定する
 8:     frame.setVisible(true); // ウィンドウを表示する
    }
}
```

このソースコード片に新しく登場するクラスはいずれも Java 標準ライブラリのクラスである．以下のクラスを import する文を記述すること．

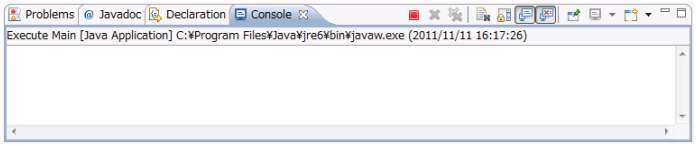

図 19: 実行中の「コンソール」タブ．赤い四角のアイコンをクリックするとプログラムを強制終了する．

- javax.swing.SwingUtilities
- javax.swing.JFrame
- javax.swing.JPanel
- java.awt.Dimension

このプログラムは，ウィンドウを表示するだけの最小限のプログラムである．main メソッド中で呼び出している invokeLater メソッドは，引数に run() メソッドを持つオブジェクトを取り，演習で使用する GUI ライブラリである Swing のスレッドの一部として，必要な処理の実行を指示している．つまり，このプログラムはすぐに main の実行を完了し，あとは GUI ライブラリが勝手に動き続ける，という形式をとっている．

run メソッドが GUI 構築処理の本体である．まず，1 行目に登場する JFrame はウィンドウの「枠」を意味するクラスである．2 行目の setDefaultCloseOperation というメソッド呼び出しは，frame で参照されているウィンドウが閉じられたときに自動でオブジェクトを破棄するよう設定している．Java の GUI プログラミングでは，明示的にプログラムが終了したと宣言されるまでは main メソッドの実行が終了しても動き続ける（「解説 : GUI プログラムの実行の仕組み」参照）が，この設定を行っておくと，ウィンドウを閉じたときに自動的に破棄され，すべてのウィンドウを破棄した時点でプログラムの実行が終了するようになる．

3 行目で作成している JPanel オブジェクトはウィンドウの背景，すなわち通常は灰色で塗りつぶされている「何もない」領域を表現している．4 行目の frame に対する setContentPane メソッドの呼び出しは，frame が参照するウィンドウの内容に対応するのが base であるという設定を行っている．これにより，frame には，中身が灰色の領域だけの（何もない）ウィンドウとなる．

5 行目と 6 行目は，base のパネルと frame のウィンドウそれぞれに，画面表示にあたっての希望サイズ，最小サイズを設定している．new Dimension は 2 つの引数として横幅，縦幅を取り，サイズを表現した Dimension オブジェクトを作成している式である．5 行目であれば，横幅 400，縦幅 300 という設定を持った Dimension オブジェクトの参照を base パネルの setPreferredSize メソッドに引き渡し，パネルは横 400，縦 300 ピクセルというサイズが望ましい（Preferred）ことを指定している．同様に，frame に対しては setMinimumSize というメソッドを呼び，ウィンドウの最低サイズを指定している．パネルやウィンドウのサイズは，幅と高さによって決まるので，その値の組を表現する Dimension オブジェクトが値として使われている．横幅，縦幅の単位はいずれもピクセルである．もう少し大きなサイズが好みの場合は，それぞれ数値を大きくして構わない．

7 行目の frame に対する pack メソッドの呼び出しは，最終的なウィンドウのサイズやレイアウトを確定する処理である．8 行目の setVisible は，表示状態を true にする，つまりウィンドウの画面への表示を行っている．このコード例ではパネル 1 枚に占有された（つまりすべて灰色の領域で塗りつぶされた）ウィンドウを表示するが，今後，ウィンドウを作る作業では，base パネルの上にボタンや他のパネルなど，GUI 部品を積み重ねていく形になる．これらすべての処理は，7 行目の pack メソッドよりも前に行われていなくてはならない（FAQ Q16 参照）．

プログラムを実行し，中身が空のウィンドウが表示されることを確認せよ．プログラムを終了するには，ウィンドウを閉じること．演習室で twm を使っている場合は，ウィンドウ左上に表示されるタイトルバーの左端に表示される四角形のところにマウスカーソルを合わせてボタンを押し，「Kill Window」を選択する．

解説：GUI プログラムの実行の仕組み

コマンドライン（シェル）から実行するプログラムは，main から実行を開始して，main の実行が終了するとプログラムを終わる．これに対して GUI プログラムでは，ウィンドウが閉じられるまでプログラムを終了するわけにはいかない．そこで，GUI プログラムでは，ウィンドウ上でのマウス操作などを「イベント」とみなし,「イベント」の到着に応じてそれを処理する「イベントループ」形式でプログラムを実行する．

```
// イベントループを表現した擬似コード．このようなコードを実際に書くわけではない．
while (true) {
    次の GUI のイベントが到着するのを待つ;
    到着したイベントを処理する;
}
```

上記のような構造のイベントループは，この資料の例で使用する Swing ライブラリ（javax.swing パッケージのクラス群）によって提供されているので，自分で作成する必要はない．ウィンドウを画面に表示しようとした瞬間から，このイベントループを実行する**スレッド**が作成される．

スレッドとは，1 つのプログラムで 2 つ以上の処理を並行で実行する際の処理単位の名前である．スレッドごとに，現在の実行位置やローカル変数などを個別に保持する．OS が複数のプログラム（プロセス）を同時実行できるように，1 つのプログラム中では，複数のスレッドが並行で実行される．これまでのプログラミングでは特に明示していないが，プログラムの実行を開始したときに main の先頭から実行を開始するのが 1 つのスレッドである（便宜上 main スレッドと呼ぶ）．そして，ウィンドウを表示すると，GUI 用のスレッドは main とは並行してイベントループの実行を開始する．本章で作成する GUI の作成手順では，main スレッドは GUI の構築作業を介することだけを指示して実行を終了するが，プログラム全体の実行は，GUI スレッドが生き残っている間は継続される．GUI スレッドは，先に説明したようなイベントループを実行し，プログラム終了が明示的に指示されるまで（演習のプログラムでは DISPOSE_ON_CLOSE が設定されたすべてのウィンドウが閉じられるまで），無限ループを続ける．

もしウィンドウの操作が不可能な場合は，図 19 に示すように Eclipse の「コンソール (Console)」ビューの上端にある赤い四角のアイコンをクリックして，強制終了を行うこと．

7.2 ウィンドウにタイトルを表示する

ウィンドウのタイトルバーに「Lifegame」という文字列を表示する処理を記述せよ．ウィンドウに対する操作なので，JFrame オブジェクトに対するメソッド呼び出しである．

7.3 パネルを継承して盤面表示用クラス BoardView を作成する

ライフゲームの盤面を画面に表示するには，当然ながら，盤面を表示するための部品が必要である．しかし，Java の標準ライブラリには，単独でライフゲームの盤面を表示してくれるような都合のよい部品は存在しない．そこで，ライフゲームの盤面を表示する専用のパネルを作成する必要がある．

このような GUI 処理を担当するクラスを BoardView と名付けて作成することにする．lifegameパッケージにBoardViewクラスを作成せよ．ファイルには，次のように javax.swing.JPanel クラスを import し，extends 節を追記すること（クラス作成ダイアログでスーパークラスを指定してもよいし，手書きで加えてもよい）．

```
package lifegame;

import javax.swing.JPanel;

public class BoardView extends JPanel {

}
```

extends は，英語そのままの意味で「クラス BoardView は JPanel を拡張する」と読むことができる．これが Java の強力な機構の 1 つである継承の使用である．この時点で，BoardView クラスは，JPanel の機能をすべて受け継ぎ，現在のウィンドウ内部に表示されているような灰色の領域を表示する機能を持ったクラスとなっている．

7.4 BoardView のウィンドウへの配置

BoardView を画面に表示する処理を run メソッドに記述する．以下のコードを Main クラスの run メソッドの中，pack を呼び出す行の直前に追加せよ．新しいクラス java.awt.BorderLayout が登場するので，必要な import 文も忘れずに追加すること．

```
base.setLayout(new BorderLayout());      // base 上に配置する GUI 部品のルールを設定
BoardView view = new BoardView();
base.add(view, BorderLayout.CENTER);     // base の中央に view を配置する
```

この新しい命令の 1 行目では，base パネルが BorderLayout というルールに従って内部のオブジェクトをレイアウトするよう指示している．2 行目はこれまでと同様のオブジェクトの作成で，3 行目で作成した BoardView を base の中央に配置するよう指示している．BorderLayout は，指定した領域の端から端まで広げるように配置するというルールであり，NORTH であればウィンドウ上端に（一般的なアプリケーションのメニュー項目のように），EAST であればウィンドウ右端に（一般的なアプリケーションのスクロールバー

無視できる警告

BoardView が JPanel を継承した時点で，以下のような警告が「問題 (Problems)」ビューに登場する．

```
The serializable class BoardView does not declare a static final
serialVersionUID field of type long
```

BoardView の親にあたる JPanel クラスが直列化可能（serializable）であるが，その操作に必要な serialVersionUID というデータを BoardView が定義していないことが警告の原因である．

直列化というのは，Java のオブジェクトが格納しているデータをメモリ上から収集し，ディスクなどに保存可能なデータ形式に変換する操作のことである．Java が出てきた当時には，複雑な構造のオブジェクトを丸ごと保存したりネットワークに流したりできるので便利だと考えられていたが，実際にはそれほど活用されていない．本演習でも，直列化を使う機会はないので，このメッセージは無視してかまわない．

警告が気になる人は，エラーメッセージを右クリックして「Quick Fix」を選び，表示されるダイアログのメニューから「Add generated serial version ID」を指示すること．Eclipse が必要な定数定義を生成して警告を解消してくれる．

のように），それぞれ配置される．CENTER による中央配置というのは，ウィンドウのサイズに合わせて領域全体を占有するという意味である．この時点ではプログラムの実行時の見た目は変化しないので，画面に BoardView が表示されているかどうかを確認するために，次節の描画テストへと進む．

7.5 BoardView の描画テスト

BoardViewクラスにおいて，JPanelから継承している paintメソッドを以下のようにオーバーライドし，ウィンドウに直線や四角形を表示させてみよ．

```
package lifegame;

import java.awt.Graphics;
import javax.swing.JPanel;

public class BoardView extends JPanel {

    @Override
    public void paint(Graphics g) {
        super.paint(g); // JPanel の描画処理（背景塗りつぶし）

        // 直線や塗りつぶしの例
        g.drawLine(20, 30, 50, 30);
        g.drawLine(20, 40, 50, 40);
        g.drawLine(30, 20, 30, 50);
        g.drawLine(40, 20, 40, 50);
        g.fillRect(31, 31, 9, 9);
    }

}
```

メソッド宣言の 1 行上に付与されている @Override というキーワードは,「このメソッドはオーバーライドである」ことをコンパイラに知らせるためのものである．メソッド名の綴りの間違いなどにより「オーバーライドになっていないとき」に警告を発してくれる仕組みで，付けておいたほうが少しだけ安全になる．

この paint メソッドは，ウィンドウを表示したタイミングや，ウィンドウが拡大・縮小されたタイミングなどでウィンドウ側から自動的に呼び出される．そのため，自分が書いたプログラムから直接呼び出すことはない．java.awt.Graphics オブジェクトへの参照が引数としてウィンドウ側から渡されるので，このオブジェクトに対して描画用のメソッドを呼び出す処理を記述する．

メソッドの中で 1 行目に登場している “super.paint(g);” という呼び出しは，JPanel から継承した paint 機能を呼び出しており，パネルの全面をいわゆる背景色（多くの環境では灰色）で塗りつぶしている．この文よりも後ろに盤面を描く命令を書くことで，画面に自由に絵を表示することができる．また，このコードを消してしまうと，ウィンドウに他のウィンドウが重なるなどして「消えた」部分が塗りなおされなくなってしまう（自分で背景を塗りなおす命令を書く必要が出てくる）．なお，super は，一見すると変数に見えるが，スーパークラスに定義されたメソッドを呼び出すための特別なキーワードである（変数のように使おうとすると構文エラーになる）．

メソッドに登場する drawLine メソッドは引数として (x1, y1, x2, y2) の 4 つを取り，BoardView パネル内部の座標 (x1, y1) から (x2, y2) へと直線を引くメソッドである．fillRect メソッドは引数として

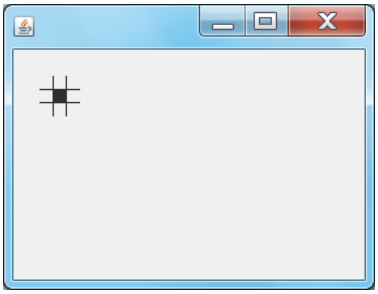

図 20: 7.5 節の時点で BoardView が正しく表示された場合の画面の例.

継承とオーバーライド

継承を使うと，親クラス（スーパークラス）からすべての機能を継承した子クラス（サブクラス）を簡単に作ることができる．そして，親クラスに宣言されたものと同名，同じ引数のメソッドを宣言することで，親クラスの機能をオーバーライド（override; 上書き，あるいは置き換え）することができる．

BoardView に作成する paint メソッドは，JPanel に元々書かれている paint メソッドをオーバーライドする形で作成している．

Java の GUI のライブラリを作った人から見ると BoardView というのは未知のクラスであるが，「BoardView オブジェクトは画面への表示機能以外は JPanel の機能を継承している」という情報があることから，言語処理系は BoardView オブジェクトを GUI に組み込むことを許している．

(x, y, w, h) の 4 つを取り，(x, y) を左上端，横幅 w，縦幅 h の四角形の範囲を塗りつぶすメソッドである．ここに記述した 5 行の命令には特に深い意味はなく，単に BoardView オブジェクトが画面に表示されていることを確認するためだけの記述である．ここまでプログラムを正しく書けていれば，実行すると，図 20 のようなウィンドウが表示される．drawLine や fillRect メソッドの動作を詳しく確認したい場合は，それぞれの引数などを変更してみて，画面の表示がどのように変化するかを試してみると良い．

7.6 盤面を描画する

BoardView クラスの paintメソッドを変更し，ライフゲーム用の盤面を描画する機能を実装せよ．描くべき盤面の仕様は，本資料の冒頭部にある「作成するプログラムの機能」「GUI による盤面の表示」の項目を確認すること．盤面のサイズを横に N 列，縦に M 行あるとすると，縦横それぞれ，$N+1$ 本，$M+1$ 本の直線によって領域を仕切ればよいことになる．絵を描くためのヒントや，利用可能なメソッドを以下に示すので，paint の内容を自分で考え，記述し，動かしてみること．

- BoardView オブジェクトに，盤面の情報である BoardModel オブジェクトへの参照を渡す必要がある．paint はウィンドウが表示された時点で外部から自動的に呼び出されるメソッドなので，これが実行されるよりも前に BoardModel を渡すようにすること．BoardView クラスには好きなようにコンストラクタやメソッドを定義できるので，方法は自由である．
- BoardModel オブジェクトからセルの状態を取り出すために isAlive(int x, int y) といったメソッドを追加すると良い．直接フィールドなどを外部から参照するのは Java では一般的な方式ではなく，また，柔軟な機能拡張が可能になるという点からもメソッドの使用が推奨される．
- 絵を描く対象であるパネル（BoardView）の大きさは，ウィンドウの拡大・縮小によって変動する．現在のパネルの幅は this.getWidth() で，高さは this.getHeight() でそれぞれ取得することができ

る（いずれも JPanel から継承しているメソッドを呼び出す）．各メソッドの戻り値を w, h とすると，$w \times h$ 個のピクセルの並びが絵を描ける領域で，左上の座標が $(0, 0)$，右下の座標が $(w-1, h-1)$ となる．なお，これらのメソッドは，パネルが画面に表示されるまでは正しい値を返さないため，たとえばコンストラクタ中で呼び出しても無意味である．paint メソッド中から使用すること．

- 実際に直線等を描くには，引数として渡される jawa.awt.Graphics オブジェクトのメソッドを使う．コード例としても既に登場している drawLine が利用可能である．メソッドの詳細情報が必要な場合は，Graphics クラスの API を参照すること．
- BoardView クラスの内部，他のメソッド宣言の外側であれば，他に作業用メソッドを自由に定義することができる．たとえば，セルの列，行の値から画面上の X，Y 座標に変換する関数を自分で定義すれば，paint メソッドの内容を簡潔に保つことができる．

コードを変更するたび，プログラムを実行しなおし，画面に表示される盤面の内容を確認することが望ましい．また，paint メソッドは，サイズが変更された場合などにも自動的に呼び出されるので，表示されたウィンドウのサイズを変更して，どのようなサイズでも正しく盤面が表示されることを確認すること．座標の計算方法をうまく定義できると，ウィンドウの中央に盤面を大きく表示できるので，工夫してみると良い．

なお，最終的なウィンドウのレイアウトで操作用のボタンが加わることになるが，盤面の描画については BoardView の全面を使って問題ない（余白を残さなくてよい）．BoardView に盤面を表示した状態で，ボタンを乗せた別のパネルを BoardView に隣接するようにウィンドウ上に配置することができる．

プログラムが正しく動作していることを確認するには，main メソッドの changeCellState 呼び出しの引数などを変更し，指定したセルが生きている状態として画面に表示されるかを確認すると良い．

7.7 ここまでのまとめ

このステップで，ウィンドウを開き，盤面の情報を表示できるようになったはずである．盤面を描くにあたっての座標計算の方法や，マウスの座標からセルの座標を計算する方法などは，ちょっとした工夫のしどころである．たとえば,「常に画面中央に，できるだけ大きく描く」といった意図や，それに合わせた座標の計算式の特性（ウィンドウのどれだけの範囲を盤面に使用できるかなど）は，レポートに記述するべき情報である．盛り込んだアイディアは，できるだけメモとして残しておくと，レポートの執筆が楽になる．

8 Next ボタン，Undo ボタンによる世代の進行と巻き戻しの実現

ウィンドウに「Next」「Undo」の 2 つのボタンを設置し，クリックするたびに盤面の状態を更新できるようにせよ．作業としては，ウィンドウにボタンを追加する処理を記述し，次いで，ボタンが押されたときに実行するべき処理を記述していくことになる．必要な Java の言語要素はここまでで一通り登場してきたので，自分で作業を進めてみること．注意すべきポイントは次の通りである．

ソースコードのお手軽バックアップ

Eclipse では，プロジェクト自体を選んでコピー＆ペーストを行うと，実はプロジェクト全体をコピーした別プロジェクトを作ってくれる．ただし，編集するファイルを間違いやすくなる（バックアップのほうを誤って編集してしまいやすい）ので，コピーしたプロジェクトの名前に日付を入れるなどしてから,「プロジェクトを閉じる（Close Project）」ことで編集対象から外しておくことが望ましい．

コマンドラインに慣れている人は，ホームディレクトリの中にある workspace というディレクトリを丸ごと zip コマンドなどで圧縮するというのが手軽かもしれない．

- ボタンは javax.swing.JButton クラスである．ボタンが押されたときに実行したい処理を記述するには，java.awt.event.ActionListener インタフェースを implements したクラスを作成し，そのオブジェクトを addActionListener で登録する必要がある．
- ウィンドウの全体を表す base パネルに直接ボタンを配置しても良いが，ボタンを適切なサイズで複数並べたい場合，FlowLayout が見た目の面で優れている．以下のコードは，base パネルの下端に，ボタンを配置するためのパネルを設置するコードである．

```
JPanel buttonPanel = new JPanel();           // ボタン用パネルを作成し
base.add(buttonPanel, BorderLayout.SOUTH);   // base の下端に配置する
buttonPanel.setLayout(new FlowLayout());     // java.awt.FlowLayout を設定
```

  上記コードに続けて，buttonPanel に JButton を追加すると，対応するボタンがウィンドウ下端に（盤面の描画範囲の外側に）buttonPanel として確保された領域に出現する．
- BoardModel の盤面の状態が更新されたとき，BoardView を描き直す処理を記述する必要がある．これには，BoardView が JPanel から継承している repaint メソッドを呼び出せばよい．
- Undo ボタンは，プログラムの実行開始直後など，巻き戻し不可能なときはボタンを無効にする必要がある．ボタンの有効・無効を切り替えるには，setEnabled メソッドを使用する．プログラムの実行開始時と，盤面の状態が変更されたとき，ボタンの状態を変更すればよい．BoardListener を作成して追加すれば，盤面の状態の変更には簡単に対応できる．

この機能が実装できたら，「Next」ボタンを 32 回程度押し，「Undo」ボタンでボタン操作とマウス操作が正しく巻き戻せるかを確認せよ．よくある誤りとして，1 回余分に巻き戻せる，あるいは巻き戻し回数が 1 回足りないという症状がある．

9　マウスによる盤面編集機能の追加

マウスのクリックで盤面の状態を編集する機能を実装せよ．具体的な処理としては，BoardView 上でマウス操作が起きたときにそのカーソルの座標を取得し，対応する盤面上の座標に変換してから BoardModel の changeCellState メソッドを呼び出す処理を記述することになる．

9.1　マウス操作の情報をメソッド呼び出しとして受け取るようにする

マウス操作は，元々，ウィンドウシステムに管理されている情報である．ユーザによるカーソル移動やクリックなどの操作が適切に解釈された結果だけが，各ソフトウェアに対して「メッセージ」という形で配布される．具体的には，ある GUI 部品の領域上にマウスカーソルが進入した（Entered），領域中をカーソルが移動した（Moved），ボタンが押された（Pressed），離された（Released），カーソルが部品の領域上から去った（Exited）という一連の操作がメッセージとして与えられる．また，ボタンが押されてすぐ離された場合はクリックした（Clicked）というメッセージが追加で与えられる．そのほか，ボタンを押してのカーソル移動（Dragged），ホイールの回転（WheelMoved）もメッセージとして存在する．各メッセージには，マウスカーソルが指している画面上のピクセルの座標や，マウスのボタンの状態，CTRL や SHIFT などの特定のキーの押下の有無などが情報として格納されている．

マウス操作の情報を受け取るには，Swing ライブラリが提供しているイベントリスナの仕組みを利用する．これは先ほど作成した BoardListener と同様のもので，マウスの操作に対応する特定の名前のメソッド，た

とえば mouseClicked や mouseEntered を定義したオブジェクトを用意し，Swing が提供している GUI 部品（パネルやボタンなど）に登録する．ウィンドウシステムから通知されたメッセージが Swing の各 GUI 部品に到着したとき，Swing はその情報をメソッド呼び出しとして登録されているオブジェクトに通知する．

マウス操作の通知を受け取るためのメソッドは，次の 3 つのインタフェースに定義されている．

- java.awt.event.MouseListener インタフェース．Entered，Pressed，Released，Exited，Clicked の 5 つの通知を受け取るためのメソッド宣言を規定している．マウスのクリックなどの基本操作に対応する場合に使用する．
- java.awt.event.MouseMotionListener インタフェース．Dragged，Moved の通知を受け取るためのメソッド群を規定している．ドラッグ操作に対応する場合に必要となる．
- java.awt.event.MouseWheelListener インタフェース．WheelMoved の通知を受け取るためのメソッドを規定する．ホイールの回転を検出したい場合は，このインタフェースを使用すればよい．本演習では，拡張機能として使用するのでない限り，特に必要ない．

BoardView オブジェクトがマウス操作の通知を受け取れるようにするには，BoardView がインタフェースを備えている旨をクラス宣言に書き加える．複数のインタフェースを記述するには，カンマで区切って指定する．以下のコードは，BoardListener を実装しているところに，さらにインタフェースを追記した例である．

```
public class BoardView extends JPanel
    implements BoardListener, MouseListener, MouseMotionListener {
    /* クラス定義は省略 */
}
```

追加した 2 つのインタフェースを実装するために，以下のようにメソッド定義をクラスに追加する必要がある．各メソッドの引数 java.awt.event.MouseEvent は，マウス操作イベントが生じたときのカーソル位置などの情報を格納しているオブジェクトである．

```
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }
    public void mouseDragged(MouseEvent e) {
    }
    public void mouseMoved(MouseEvent e) {
    }
```

この時点では各メソッドの内容が空になっているので，BoardView は通知を受け取ることができるが，受け取っても何もしない，という状態である．

メソッドの宣言ができたら，イベント通知を受け取るための登録を行う必要がある．BoardView 上で起きたイベントを BoardView 自身が受け取る場合，コンストラクタなどで，次のように自分自身を登録すればよい．

```
this.addMouseListener(this);
this.addMouseMotionListener(this);
```

ここで使用している addMouseListener，addMouseMotionListener は，いずれも JPanel から継承しているメソッドである．

ここまで作成してきたコードが正しく動いていることを確認するには，以下のような実験用コードを mousePressed メソッドなどに書き加えてからプログラムを実行し，パネルをクリックして，コンソールにマウスの情報が書き出されること，また盤面の状態が更新されることを確認してみると良い．

```
public void mousePressed(MouseEvent e) {
    // 標準エラー出力にクリックされた座標を書き出す
    System.err.println("Pressed: " + e.getX() + ", " + e.getY());

    // (1, 1) のセルの状態を反転させる
    model.changeCellState(1, 1);
}
```

9.2 マウス操作の情報を解釈する処理を作成する

マウス操作の通知を受け取れるようになったら，作成するプログラムの機能のうち「マウスによる盤面の編集」の項目に書かれた仕様を達成するために，マウスのボタン押下，ドラッグ操作に対応するための処理を記述せよ．これらの処理は，整理すると，次の 2 つの処理である．

- マウスのボタンが押されたとき，カーソルの座標がセル (x, y) の座標の範囲内であれば，そのセルの状態を変更する．
- マウスがドラッグされたとき，カーソルの座標がセル (x, y) に対応していて，かつ，直前のイベントが同一セル内でのマウスボタンの押下とドラッグ移動のどちらでもなければ，そのセルの状態を変更する．

両者とも，マウスの座標からセルの座標に変換する操作が必要である．前のステップで作成した paint の実装を見直し，マウスカーソルの座標とセルの座標の関係式から，計算方法を考えること．また，ドラッグの実装では，適宜フィールドを用いて直前の状態を記憶しておく必要がある．

処理が実装できたら，盤面の様々な場所でクリック，ドラッグ操作を行い，セルの状態が正しく変更されることを確認せよ．

9.3 起動時の盤面を空にする

盤面の操作が正しく実行できるようになれば，プログラムの実行開始時に行っている changeCellState の呼び出しなどが不要になる．また，ModelPrinter も不要になる．main メソッド中の該当処理を取り除き，起動直後の盤面を空にしておくこと．

9.4 ここまでのまとめ

このステップで，GUI からのライフゲームの制御がほぼ完成する．マウス操作に対応するためのメソッドは多数登場したが，必須機能を使うために必要になるのは Pressed と Dragged の 2 つのイベントだけである．マウス操作は，1 回のクリックが，Pressed，Released と Clicked の 3 つのイベントとして認識されるといったように，使うには慣れが必要である．各メソッド呼び出しの発生順序や引数として渡される `MouseEvent` の中身を詳しく知りたい場合は，`System.err.println` などを用いて情報を書き出してみること．

10 新しいウィンドウを開く機能の実現

「New Game」ボタンを追加し，押すたびに空の盤面を持つ新しいウィンドウを開く処理を実現せよ．空の盤面を作り新しいウィンドウを開くという一連の処理は，既にプログラム中に存在している．`static` フィールドによるデータの共有などを行っていなければ，`new` によって作られた盤面がそれぞれのウィンドウで独立に動作するはずである．

11 拡張機能の実装

ライフゲームの必須機能の実装が終了した後は，演習の残り時間が許す範囲で拡張機能の実装に取り組むこと．拡張機能は，独自性，技術的難易度，操作性が評価の対象となる．本章は，様々な拡張機能の例と，その実装のためのヒント集である．

ここに記載している以外の，独自の拡張を施してもよい．ただし，機能拡張において，ライフゲームの本来の機能に無関係な拡張は評価対象外となることがある．

11.1 アニメーション機能

機能の概要 アニメーション表示を行うための「Auto」ボタンをウィンドウに追加し，「Auto」ボタンを押すと，0.5 秒につき 1 世代程度の速度で，自動的に盤面を更新するようにする．

自動更新中も，ウィンドウの移動などの GUI 操作は受け付ける必要があることに注意すること．また，ウィンドウが閉じられたとき，スレッドを終了するのを忘れないこと．スレッドを終了しないと，ウィンドウをすべて閉じてもプログラムの実行が終了しなくなる．

実装のヒント ウィンドウの操作など，他の機能の実行を妨害しないようにアニメーション表示を行うには，スレッドを使って，一定時間おきに処理を実行する仕組みが必要となる．スレッドの効果的な使い方は「Effective Java」（ピアソン・エディケーション，2008）に詳しく書かれている．

ユーザのマウス操作と自動実行が同時に呼び出されると，データの一貫性が失われる可能性があるので，`BoardModel` クラスの各編集メソッドは，`synchronized` キーワードを指定する必要がある．

11.2 テキストファイルの保存，読み込み

機能の概要 メニュー [File]-[Save As ...] を追加し，盤面の状態を任意の名前のテキストファイルとして保存できるようにする．また，メニュー [File]-[Open ...] を追加し，盤面の状態を保存したテキストファイルを読み込めるようにする．

実装のヒント　ファイルの読み書きには，ファイルそのものに対応するクラス `File` と，読み書き用のクラス `FileReader`，`FileWriter` を用いる．`Reader`，`Writer` 系のクラスは，読み書きが終了した後に `close` メソッドを呼ばなくてはならない．演習室の FreeBSD 環境では，OS のファイル管理の仕組み上，エラーが起きにくいが，Windows 上では開いたままのファイルは書き込み禁止になるので，ファイルを書き出した後，さらに上書き保存しようとしたときに実行時エラーが発生する．`Reader` や `Writer` でファイルを開いたのなら，必ず閉じること．

ファイルの形式については自分で考え，レポートにもそれを記載すること．読み込みを指示されたファイルの形式が想定と違う（無関係なファイルを与えられた）場合への対応は，工夫のしどころである．また，ファイルを読み込んだり保存したときに巻き戻し用の履歴をどう扱うかも，設計上考えるべき点である．

なお，ファイル入出力については，読み込もうとしたファイルが存在しない，あるいは書きこもうとしたファイルがロックされているなどの理由で処理が失敗することがある．このとき，`IOException` という例外が発生するので，`try-catch` 文を使って例外を捕捉し，ユーザへの通知などを行うこと（`try-catch` 文を用いた例外処理を記述しない限りコンパイルエラーとなる）．エラーが起きたときに出されるメッセージの適切さも，評価の対象となる．

11.3　盤面のサイズ変更

機能の概要　数値を入力するダイアログを準備し，異なるサイズの盤面を作成できるようにする．

実装のヒント　盤面のサイズ等をユーザに選択させる GUI を作成する場合は，`JDialog` クラスが便利である．`JDialog` は，`JFrame` とほとんど同じクラスであるが，`setModalExclusionType` メソッドによって，表示中にアプリケーションの他のウィンドウへのアクセスを防止する設定を行うことができる．

盤面のサイズ等の数値を入力する場面では，その値が妥当な範囲（画面上に表示可能な範囲）になるよう，適切に範囲を限定しておくことが望ましい．

11.4　生きているセル数のモニタリング

機能の概要　盤面で生きているセルの数を世代ごとに記録していき，セルの生存状況の時系列での変化をグラフ等で閲覧できるようにする．

実装のヒント　`BoardListener` を使えば盤面の変化を認識できるので，数値の系列を記憶し，可視化する方法を考えればよい．`BoardView` と同様に，`JPanel` を継承してグラフ専用の表示領域を作ることができる．

11.5　パターンのコピー＆ペースト

機能の概要　SHIFT キーを押しながらのドラッグ操作などで範囲を選択し，メニュー [Edit]-[Copy] からパターンを記憶できるようにし，メニュー [Edit]-[Paste] から貼り付けられるようにする．また，著名なセルの配置パターンをメニュー [Edit]-[Glider] などとして選択し，盤面の指定位置に配置することを可能にする．

実装のヒント　パターンの配置を行うには，`changeCellState` のようにセルの状態を反転させるのではなく，パターン定義に従ってそのまま値を変更するためのメソッドが必要である．

配置のためのユーザインタフェースも重要である．マウスカーソル位置に対してどの範囲にパターンが配置されるを表示する方法については，工夫の余地が大きい．また，パターンの大きさに合わせて，盤面の外にはみ出す場合は置かせない，あるいはうまく盤面に納めるよう調整するといった対応も考えられる．

「ライフゲームの本来の機能に無関係な拡張」の例

機能拡張として認めない「ライフゲームの本来の機能に無関係な拡張」については，残念ながら明快な基準が指定できないが，何でも「機能拡張」と言い張ることを予防するためのガイドラインである．

具体例を挙げておくと，まず,「プログラム起動時に “Hello, World!” というメッセージを出力する機能」や「プログラム起動時に盤面のサイズをコンソールに出力する機能」は，前者はメッセージの内容が利用者にとって意味がなく，また，後者は一般にはデバッグ用と思われるので不適切である．

極端な事例の 1 つであるが，過去の Java 演習において，プログラムの課題「住所録」に対して，機能拡張として「テキストエディタ」（長文テキストの編集機能）を付けてきた事例があった．このテキストエディタにはファイル読み書きなどのいくつかの機能が付いていたが，データは住所録とは無関係に管理されており，住所録の機能やデータと相互に連携することはまったくない（たまたまテキストエディタと住所録が同じウィンドウに同居しているだけ）という状態であった．

新しい機能拡張を実現しようという場合は，利用者にとっての有用性，利便性などをよく考えてから取り掛かること．また，上記のような事例に近いと思う場合は，事前に教員に相談すること．

12 レポートの作成と提出

プログラムが完成したら，プログラムについて記述したレポートを作成し，提出すること．

12.1 プログラムの動作確認

提出前に，プログラムのすべての機能を連続的に実行し，動作を確認せよ．動作確認を行いながら，スクリーンショットの撮影を行っておくと，レポート用の機能一覧と操作説明にそのまま使用できる．動作確認において注意すべき項目には，次のようなものがある．

- 起動時に表示されるのが空の盤面であること．すべての操作用ボタンが表示されていること．また，「Undo」ボタンが無効であること．
- 盤面のセルの境界線上や，セルが描かれていない領域でマウスのクリックを行ったとき，想定どおりの振る舞いをすること．座標計算が正しくない場合，ウィンドウ右下端付近で状態を変更するべきセルを誤る場合がある．
- 32 世代以上盤面を進めても問題が起きないこと．
- 「Undo」ボタンを 32 回押したらボタンが無効になり，それより前に戻れないこと．
- マウスのドラッグ操作で，ボタンを押したままウィンドウの外やボタンの上などにカーソルが移動しても問題が起きないこと．
- 複数のウィンドウを開いた状態でも盤面の編集や世代の進行，巻き戻しが問題なく行えること．

ファイル等のシステムリソースの解放忘れに起因するエラーや，プログラム実行中に実行時例外が生じた場合，たとえば `ArrayIndexOutOfBoundsException` や `NullPointerException` などの `RuntimeException` のサブクラスが発生するプログラムは受理しない．動作確認時は，デバッガを使って例外を捕捉するため，`try-catch` によって例外を捕捉するという対策は無効である．提出前にプログラムの誤りを修正しておくこと．不十分な動作確認のまま提出し，提出後に問題が発覚した場合は，減点の対象となる．

12.2 プログラムの提出用アーカイブファイルを作成する

Eclipse プロジェクトの全ファイルを格納した zip アーカイブファイルをレポートとともに提出する．提出用のファイル名は，プロジェクト名と同一，すなわち `lifegame09Bxxxxx.zip` のように学籍番号を含んだものとする．

zip ファイルを作成するにあたって，まず，プロジェクト名が適切（学籍番号を含んだもの）であることを確認せよ．変更する場合は，パッケージ・エクスプローラからプロジェクトを選択した状態で,「ファイル (File)」メニューの「名前変更 (Rename)」を実行する．

プロジェクト名が正しく付与されていれば，続いて，メニュー「ファイル (File)」「エクスポート (Export)」によってエクスポートダイアログを開く．ダイアログでは，まずファイルの種類を質問されるので,「一般 (General)-アーカイブ・ファイル (Archive File)」を選ぶ．続いて，詳細を設定する画面で，出力ファイル名（To archive file）を指定して「Finish」ボタンを押せばよい．他の設定がデフォルトのままであれば，プロジェクトの全ファイルを zip 形式で書き出してくれる．

作成後は必ずアーカイブファイル内に全ソースコードが格納されていることを確認すること．一部ファイルが格納されていないアーカイブなどが提出された場合は，提出物不備による再提出を求める．

12.3 レポートの要件

レポートは PDF ファイル，ページサイズは A4，1 段組みの記載とする．ファイル名は `09Bxxxxx.pdf` というように，学籍番号そのものに拡張子 `.pdf` を付与したものとする [10]．レポートに記載する内容は次の通りである．

- 表紙（1 ページ目）に授業科目名，担当教員名，レポートの提出日付，提出者名，学籍番号，内容に不備があった場合の連絡先となるメールアドレスを表示すること．いずれも書類としての必要事項である．
- 作成したプログラムの操作方法と機能を説明せよ．本演習課題で指定しているプログラムが満たすべき条件では，盤面の大きさや，ドラッグ操作の巻き戻しの扱いなどに設計の自由度がある．そのため，課題としての要件ではなく，作成したプログラムが持つ機能を説明すること．

  この記述では網羅性が重要である．コマンドの実行に関する制限（特定状況でのみ使用できるなどの特徴）やエラー発生時の動作などを含めて，プログラムを操作して観測可能な振る舞いを解説すること．拡張機能に関しては，操作方法や動作確認の方法が説明されていない場合は，採点の対象外になる．
- プログラムを構成する各クラスが，どの機能の実現を担当しているか，対応表を記述せよ．また，役割分担の方針と，実際に出来上がったものがその方針をどの程度達成しているかを説明せよ．
- プログラムの各機能の実現方法の詳細を解説せよ．少なくとも以下の内容を解説すること．
  - 盤面の状態の更新に連動して更新される画面の表示項目と，その更新方法．
  - 巻き戻しのための盤面の状態の記録方法（データ構造とその操作），巻き戻し可能かどうかを判定する方法．
  - 盤面の描画において，セルの境界線の位置を計算する方法・計算式．
  - マウスカーソルの座標から対応するセルの座標を計算する方法・計算式．
  - 新しいウィンドウを開く方法．

[10] この条件は，採点前にファイルを機械的に整理する段階で，他人とファイル名が衝突し，採点対象から外れるという事故を避けるために非常に重要である．ファイル名には必ず半角英数字を使うこと．

– 拡張機能がある場合は，それぞれの実装方法．

- Java プログラミングの講義・演習で学習した内容を整理，報告せよ．整理する方法，観点は自由である．たとえば演習前から知っていたことは何であり，授業資料や参考書から修得したことは何か．演習で試すまで理解できなかった概念はあるか．また，演習を終えた状態の知識で新たに同一機能のプログラムを書き起こすとしたら，作業手順やプログラムの構造はどのように変えられるだろうか．自分で内部構造などに工夫を凝らした場合は，それは効果的だったと考えるか．まだプログラムの構造に改善可能な点があるか．Java プログラミングを自分で他人に説明するとしたら，どこを特徴として挙げるか．講義・演習資料以外で使用した参考書や文献のうち，何からどのような知識を獲得したか．自分なりの考えをまとめ，記述せよ．

レポートが完成したと思ったら，提出前に必ず PDF を自分で開き，一度は「本文」を自分で読み直して確認すること．採点の際は，文章のところを中心に読み進めていくので，本文中に解説のないデータの列挙やソースコードの列挙などは意味がない．たとえば，図表は単に示すだけでなく，その読み方や，読み取れる重要な情報に関する説明が必要である．以下にチェックリストを示す．

- 誤字脱字，文字化け等で意味のとれない表現が存在しない
- 文字列やソースコード，図表のページからの飛び出しがない
- 引用したソースコードが正しくインデントされている
- 図表がすべて本文中で参照されている
- A4 用紙と同じ大きさで表示したときに図表の文字が十分に視認できる大きさである

なお，例年，破損したファイルや，別科目のレポートが提出されるなどの提出トラブルも多い．このような提出遅延であっても減点対象となるため，提出後，ファイルが正しく提出されていることを CLE 上であらためて確認すること．

12.4 ファイルの提出とその後について

提出されたプログラムは，教員が動作確認を行う．提出物が条件を満たしていない場合や，必須機能が動作しないなどの問題がある場合にはレポートを受理せず返送するので，速やかに問題を修正し，再提出すること．本演習課題の仕様の不備に起因する振舞いや，表示文字列の誤りなどの軽微な欠陥については，減点対象となることはあるが，レポートとしては受理する．

動作確認が完了すると，レポートを受理した旨を CLE 上で通知する．この通知は，レポート締切後，2〜3 週間をめどに行う予定である．

レポート内にソースコードは不要？

本演習に限っていうと（科目により方針が違うので注意すること），ソースコードは別途提出し，動作確認を行うものなので，レポート内部に全文を貼り付ける必要はない．特に説明したいポイントがある場合にだけ，その部分を限定で引用すること．Java の命令の 1 行 1 行は非常に単純なものなので，そのソースコードの目的（プログラム中での役割）は何なのか，何をどのくらい考えた結果その内容に至ったのか，といった情報のほうが重要である．

単にクラス名やメソッド名のリストや，プログラムのリストが貼り付けられているだけというレポートでは,「そこに何を見てほしいのか」が記載されていないので，評価は低くなる．

レポートの提出が遅れる場合は，締切までに理由と提出見込み時期をメールで連絡すること．また，締切後にレポートの提出や差し替えを行った場合は，見落としを避けるために，教員までメールで連絡を行うこと．

付録：過去の Java 演習における Q & A

Java の言語に関する話

Q1. コンパイルエラーの原因は？

演習でよく見かける例は，次のようなものである．

- 予約語，メソッドや変数の綴りが間違っている．
- 適切なクラスの `import` が足りない．型名に対して赤線が引かれているときは，ほとんどの場合，これが原因である．
- クラス名とファイル名が一致していない．ファイル名を変更せずにエディタでクラス名だけを変更するとエラーになる．
- 宣言・文の記述位置が正しくない．クラスの中にフィールドやメソッドを記述し，メソッドの中に命令を記述するという階層構造を守ること．右ブレース（`}`）の過不足が原因であることも多いので，インデント（字下げ）によって対応関係を明示することが誤りの予防につながる．
- 全角空白やその他の制御文字，異なる文字コードのデータがソースコードに混入してしまっている．漢字入力モードのまま字下げを行ったり，PDF からのコピー＆ペーストによって発生することが多い．このときは，これらの文字を削除すればよい．
- ピリオドの左側に，オブジェクトの参照を格納した変数を指定するべきところを，クラス名を指定している．クラス名を指定して呼び出せるのは `static` メソッドのみである．

1 つの構文誤りが複数のエラーにつながることも多い．コンパイラはプログラムの先頭から順にプログラムを解釈していくので，ファイルの先頭付近から順に問題を確認していくと効率的に対処できる．

Q2. パッケージって意味があるの？

自作したプログラムのクラスをパッケージに入れておくことで，Eclipse の検索機能やエラーメッセージに登場したクラス名が自分のプログラムに所属するものかどうかをすぐに識別できるようになるので，劇的にというほどではないが開発作業が楽になる．

Java の標準ライブラリでは，全部で（外部から見えないものも含めて）15000 個程度のクラスが存在しており，`java.awt.List`（項目を一覧表示する GUI 部品）と `java.util.List`（データ構造のリスト）のようにパッケージ名まで含めて初めて意味が判別するクラスもある．

Q3. クラス，オブジェクトとは何か？

オブジェクトの機能を定義する，プログラムの部品単位である．英単語の本来の意味は，Collins COBUILD Advanced Lerner's English Dictionary 2006 には，次のように記載されている．

```
A class of things is a group of them with similar characteristics.
```

C 言語からプログラミングを学習している人にとっては，変数に加えて関数も要素として所有できる構造体（`struct`）の一種だと考えると分かりやすいかもしれない．この考え方に照らすと，クラスとは構造体の定義そのものであり，オブジェクトは実際にメモリ上に作られた構造体の記憶領域である．構造体を 1 つ定

義してから malloc によってデータ領域を複数確保するのと同様に，クラスを定義するとオブジェクトは複数（new を実行した回数だけ）作成することができる．

なお,「オブジェクト」のことを「インスタンス」と呼ぶことがある．インスタンスとは事例の意味で，クラス X のオブジェクトのことをクラス X のインスタンスとも表現する．また，オブジェクトを作ることを，インスタンス化すると表現することもある．

Q4. メソッドとは何か？

メソッドは，オブジェクトに関連付けられた関数である．オブジェクトにとって「自分の」フィールドの値と引数を使って，メソッド内部に書かれた命令を実行していく．一方，static メソッドは，クラスに直接所属する関数で，引数を使ってメソッド内部に書かれた命令を実行する．

メソッドの内部には，Java の基本命令は C 言語と似ており,「できること」がそれほど大きく増えたわけではない．主な命令を以下に示す．

- 四則演算や論理演算を実行し，結果を変数に代入する．
- if 文，switch 文により条件分岐を行う．
- for 文，while 文によりループを実行する．
- new によりオブジェクトを作成し，変数に代入する．
- 変数に格納されているオブジェクトに対してメソッドを呼び出す．
- return 文により，何らかの値を戻り値として返す．

C 言語との大きな違いは，static ではないメソッドは，必ずオブジェクトに所属していることである．あるメソッドの実行中は，ローカル変数以外に，以下の変数にアクセスすることができる．

- メソッドの引数．引数はすべて値渡しである．
- 自分自身のオブジェクト参照（this）．
- 自分自身のクラスに宣言されたフィールド．
- スーパークラスから継承した protected および public フィールド．
- 任意のクラスの public static フィールド．System クラスの out フィールドが代表である．

また，これらの変数を使って，次の呼び出しを行うことができる．

- アクセス可能な変数に格納されたオブジェクトに対する public メソッド．ただし，その変数が親クラスの型であれば protected メソッドを，自分自身のクラスであれば protected と private メソッドも呼び出せる．
- 任意のクラスの public コンストラクタ．
- 任意のクラスの public static メソッド．Math クラスが提供する数学関数などが該当する．

Java では，参照できる変数がオブジェクトによって決まり，参照できる変数によって使えるメソッドが決まるので,「現在，処理を実行中のオブジェクト」がどれであるかを意識するとプログラムが書きやすくなる．

Q5. メソッド，フィールドが使用できる条件は何？

対象のメソッド，フィールドが static かどうかで結果が異なる．ルールは次のようになる．

- static メソッド，フィールドを使用する場合は，ピリオドの左側はクラス名である．たとえば数学関数は Math クラスの static メソッドなので，Math.sin(x) のように呼び出す．また，標準出力は System クラスの out という static フィールドであるため，System.out という形で参照する．
- static ではないメソッドやフィールドを使用する場合は，ピリオドの左側はオブジェクト参照である．オブジェクト参照は，ローカル変数やフィールドであることが多い．たとえば System.out.println() というメソッド呼び出しは，out フィールドの参照によって得られた標準出力オブジェクト（PrintStream クラス）のメソッドを呼び出している．

  ピリオドの左側は，正しいオブジェクトさえ参照していれば，メソッドの戻り値などをそのまま使うこともできる．たとえば，整数 x を文字列に変換した結果の長さを Integer.toString(x).length() というように計算することも可能である．
- オブジェクトが自分自身のメソッドやフィールドを使用する場合は，ピリオドの左側には予約語 this を記述するが，this. 自体を省略可能である．

ピリオドの左側はそのメソッドやフィールドが所属するオブジェクトである，static メソッドとフィールドはクラス定義自体に直接所属すると認識しておくと良い．

Q6. オブジェクト参照とは何か？

オブジェクトが new によってメモリ上に作成されたとき，そのオブジェクトへの「参照」が new 全体の結果として返され，変数に格納される．オブジェクトへの参照は C 言語におけるポインタとほぼ同一で，代入文やメソッド呼び出しの引数の受け渡しによって参照をコピーしても，オブジェクト自体の数が増えることはない．ポインタとの違いは，加減算などの操作ができないこと，互換性のないクラスのオブジェクトは格納できないことである．

Q7. コンストラクタとは何か？

キーワード new が使われたタイミングで実行される，特殊なメソッドである．オブジェクトがメモリ領域に確保され，他のメソッドが呼ばれる前に実行しなければならない処理を確実に完了するために用いられる．

C 言語との対応関係で考えると，構造体のためのメモリ領域を malloc で確保する作業と，コンストラクタを実行してメモリ領域に適切な値を入れることがまとめて new というキーワードで表現されていることになる．

Q8. フィールドとローカル変数の違いは？

メソッドの中に宣言された変数は，ローカル変数となる．所属するスコープ（変数宣言を囲む { から } までの区間）でのみ有効で，その範囲外に出ると消滅する一時的な変数である．

クラスの中，メソッドの外に宣言した変数は，フィールドとなる．コンストラクタで初期値を代入され，それ以降のメソッド呼び出しで自由に参照や値の書き換えを行うことができる．フィールドはオブジェクトごとに確保されるのが特徴で，たとえば BoardModel オブジェクトを 2 個作成すれば，サイズが異なる 2 つの盤面を同時に操作することも可能である．

フィールドに static と付与された場合，そのフィールドはオブジェクトに所属せず，クラス定義に対して 1 つだけの記憶領域を持つ．System.out や System.err がこれに該当する．

Q9. 互いに呼び出し関係のないメソッド間での値のやりとりはどうしたらよいのか？

1 つのオブジェクト内部であれば，フィールド変数を使ってやり取りを行うことになる．クラス内部でのやり取りなら，private 修飾子によってクラス外からは見えなくなるので，良い選択肢である．

クラス間でやり取りする場合，参照したいデータを受け渡すメソッド（getX() や setX()）をどちらかのクラスに持たせて，メソッド呼び出しによって値を届けることになる．

Q10. エラーメッセージはどのように分析したらよいのか？

プログラムの実行中にエラーが発生すると，次のようなメッセージが標準エラー出力に出てくる．

```
Exception in thread "AWT-EventQueue-0" java.lang.ArrayIndexOutOfBoundsException: -1
    at lifegame.BoardModel.changeCellState(BoardModel.java:33)
    at lifegame.BoardView.mouseDragged(BoardView.java:44)
    at java.awt.Component.processMouseMotionEvent(Component.java:6156)
    at javax.swing.JComponent.processMouseMotionEvent(JComponent.java:3285)
    at java.awt.Component.processEvent(Component.java:5877)
  ...
```

このメッセージは，1 行目がエラーの原因を，2 行目以降がエラーの発生位置を示している．

1 行目に出てくる例外の名前（ここでは java.lang.ArrayIndexOutOfBoundsException）から，配列の範囲外を操作しようとしたことが示されている（例外名で検索をかけると，たいてい解説の Web サイトが見つかる）．右端の “-1” は，何らかの配列 a に対して，a[-1] となるようなアクセスを行ったことを示している．

メッセージの 2 行目が，その命令を実行したクラス名，メソッド名，ソースファイル名，行番号である．BoardModel クラスの changeCellState メソッド，BoardModel.java ファイルの 33 行目が，問題の処理が行われている場所である．3 行目以降は，そのメソッドが呼び出された状況を示しており，mouseDragged の実行中，BoardView.java の 44 行目から changeCellState が呼び出されたことを示している．多くのエラーでは，エラー位置情報の上から 2〜3 行を確認すれば，状況をつかむことができる．

Q11. NullPointerException とは？

オブジェクトに対して呼び出すべきメソッドやフィールドを，オブジェクトへの参照がない状態を示す値 null に対して実行した場合に発生する実行時エラーの一種である．プログラミング D の演習の範囲に限って言うと，オブジェクトを格納するための変数は作成したが new を呼び出していないといった初期化忘れが原因であることが多い．

Q12. 変数に値を代入しているはずなのに効果がないのはなぜ？

この現象が起きるときは，だいたい 2 つの間違いパターンがある．1 つは，フィールドと同名のローカル変数を使っており，フィールドに代入するつもりがローカル変数に代入しているという場合．Eclipse では，変数が青い字で表示されていればフィールド，黒い字で表示されていればローカル変数なので，それで確認すると良い．

もう 1 つのパターンは，オブジェクトを 2 つ作ってしまっている場合である．たとえば，BoardModel オブジェクトが 2 つ存在していて，画面の表示に使っているオブジェクトと，内部で操作しているオブジェクトが違うものになっていると，いくらマウスで操作しても，画面には反映されないということが起きる．この場合は，どこかに余分な new があるはずなので，それを削り，かわりに既存のオブジェクトを渡すようにする．

Q13. boolean 型とは？

boolean 型は真偽値を表現するデータ型である．boolean 型の定数（リテラル）は true と false の 2 つである．また，==，!=や大小比較の二項演算はすべて boolean 型の値を返す．boolean 型の 2 つの値が与えられたとき，&&で論理積 (AND) を，||で論理和 (OR) を計算できる．また，単項演算子 ! によって否定 (NOT) を得ることもできる．以下は，値を代入する例である．

```
boolean dataAvailable = true;
boolean done = false;
boolean continueLoop = !done && dataAvailable;
```

Java においては，if 文や for 文などの条件節に記述できるのは boolean 型の値となる式だけに限定されている．以下は，ある整数型の変数 x が 0 か 1 のどちらかであることを確認する例である．

```
if (x == 0 || x == 1) { ... } // OK.  x == 0 の真偽値と x == 1 の真偽値の論理和.
if (x == 0 || 1) { ... }      // エラー．論理和の右側が整数値になっている．
```

boolean 型の変数を使うと，次のような記述も可能である．

```
boolean isValidRange = (x >= 0) && (x < 10); // 論理積の結果を変数に代入.
if (isValidRange) { ... }   // OK.  変数に格納された真偽値を使って条件分岐する.
```

boolean 型に対しても比較演算は可能であるが，x == true は x と等価，x == false は !x と等価である．

Q14. List の要素型としてプリミティブ (boolean) は使えないのに，その配列が使えるのはなぜ？

プリミティブ型の配列はオブジェクトの一種であるから，というのが答えになる．boolean x = y; というプリミティブ型の代入文は「値のコピー」を意味しているが，boolean[] x = y; という配列の代入文は，配列 y の中身が x にコピーされるのではなく，y と x が同じ配列を指すようになるだけである．

Q15. 二次元配列をコピーする方法は？

二次元配列のコピーを行うための単純なコードは，二次元配列全体を新たに new で作成して二重ループを使って逐次的に内容をコピーするというものになる．

図 21 に示すように配列への参照そのものを代入してもコピーができないというのは理解しやすいと思うが，二次元配列は「一次元配列への参照を要素とする配列」である．そのため，clone メソッドや arraycopy メソッドを使っても，得られるのは図 22 のような状態である．この状態では，各要素を格納した配列自体が共有されているため，本来の二次元配列の複製という目的にはまったく適さない．一部インターネットの解説サイトでは図 21 を浅いコピー，図 22 を深いコピーと表現している事例を見かけたことがあるが，正しくは図 22 が浅いコピー，図 23 が深いコピーと呼ばれるものである．図 21 は単なる参照の受け渡しであって，そもそもオブジェクトのコピーではない．

Q16. ウィンドウを作る処理を書き換えたのにウィンドウの見た目が変わらないのはなぜ？

ウィンドウのレイアウトは，pack メソッドによって確定される．pack 後にウィンドウに追加で部品を配置しても，画面には反映されないので，メソッド呼び出しの順序を確認すること．

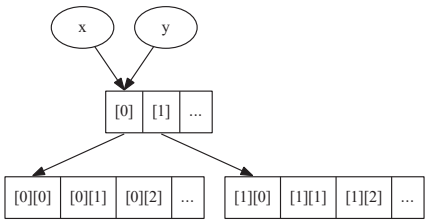

図 21: `x=y;` という代入文の場合

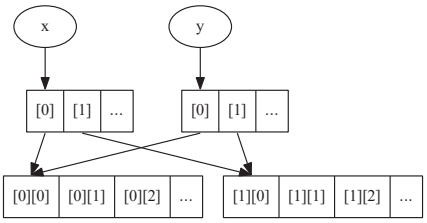

図 22: `clone` または `arraycopy` の場合（浅いコピー）

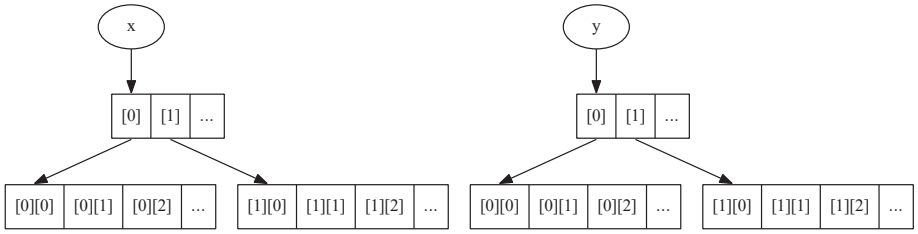

図 23: 新しい二次元配列を `new` で作り中身を二重ループでコピーした場合（深いコピー）

Q17. メソッドの戻り値として，整数の組 x, y などを返すことはできない？

Java では，メソッドの戻り値は最大でも 1 つと限定されており，値の組を直接返すことはできない．対策としては，以下の 3 つの方法がある．

- 座標を管理する `Point` オブジェクトのように，値の組を表現するオブジェクトを作成し，それを引数として渡す．ただし，頻繁に行う計算でこの方法を用いると，目に見えて実行速度が遅くなる場合がある．
- `getX()` と `getY()` のように値を個別に返すメソッドを作る．値の計算を行うメソッドと結果の参照を行うメソッドは分けるべき，という設計の流派もあるので，それほどおかしな方法ではない．
- 値を計算したオブジェクトから，値を必要とするオブジェクトへとメソッド呼出しを行うようにする．たとえば「座標 `(x, y)` がクリックされた」ことを `pointClicked(int x, int y)` というメソッド呼び出しとして伝える．呼出し関係の主従が逆転するので，使える場面は限られるが，GUI のイベント通知の仕組みがこの一例といえる．

Q18. メンバ関数，仮想関数，メンバ変数，インスタンス変数というのは何？

オブジェクト指向プログラミングは，複数のプログラミング言語によって実現されてきた経緯から，同じようなものを指すために複数の言葉が使われている．

メンバ関数（Member Function）は C++ 系の用語で，Java ではメソッドに対応する．仮想関数（Virtual Function）は，呼び出し関係がコンパイル時ではなく実行時に決まる関数のことで，Java ではコンストラクタと `static` メソッドを除く通常のメソッドが仮想関数である．

メンバ変数（Member Variable）とインスタンス変数（Instance Variable）は，いずれも Java ではフィールドのことを指す．

プログラミングスタイルに関する話

Q19. 変数，メソッドに，役割に準じた英語っぽい名前をつけるにはどうしたらいい？

プログラムで使われる動詞については「主な動詞」が限られている．具体的には，add, check, clear, close, find, get, generate, has, init, is, load, print, process, read, run, set, update, write などである．ライブラリに登場するメソッドの名前を確認してみると良い．

名詞，形容詞は，辞書を引いて調べるのが無難である．ライフゲームの場合，Cell, Board, Mouse, Cursor, Position, Width, Height といった単語が主な登場人物となるだろう．技術用語の英語表現については，IT 用語辞典 [11] などが便利である．

これら動詞や名詞の単語を組み合わせてメソッド名などを付けていくことになるが，重要なのは，プログラム全体で一貫したものを使うことである．ビジネスアプリケーションの業界では，たとえば “getKanriNumber”（管理番号を返す）のようなローマ字を含む表記も使われている．

より一般的な場合については，Steve McConnell「Code Complete 第 2 版（日経 BP ソフトプレス，2005，日本語版は上下巻構成）」で詳しく述べられている．

Q20. ソースコード中に書くコメントは日本語で書かないほうがいい？

Eclipse は日本語にもきちんと対応しているので，日本語でも英語でもよい．文字化けしないように，全ファイルの文字コードをあらかじめ統一しておくこと（1.3 節参照）．

演習に関する話

Q21. Eclipse や Firefox がクラッシュして，起動できなくなった場合はどうすればいい？

Eclipse は `workspace/.metadata` ディレクトリに `.lock` ファイルを，Firefox の場合は `.mozilla` ディレクトリ以下に作られるプロファイルディレクトリに `parent.lock` ファイルを作って二重起動を判定しているので，該当するファイルを削除する．ピリオドから始まるファイル名，ディレクトリ名は `ls -a` でないと表示されないことに注意．

Q22. 日本語が入力できないときはどうすればいい？

日本語入力の mozc は，`mozc_server_stop` によっていったん停止し，`mozc_server_start` で起動し直す．日本語入力ができない状態の Firefox などは一度再起動しないと日本語入力が可能にならない．再起動を避けたい場合は，別にテキストエディタを起動して文章を書いてからコピー＆ペーストすることになる．

`mozc` を再起動してもうまくいかない場合は， `mozc` 停止中に `.mozc` ディレクトリ（`mozc` の変換結果の学習データなどが記録される）を削除してから `mozc` を起動し直すと復活する場合がある．

Q23. Eclipse を使うのはなぜ？

Java はパッケージ宣言，クラス宣言，`import` 文などの「決まりごと」が多い言語の 1 つであり，開発環境のサポートがあったほうが開発の効率が大幅に向上するためである．また，プロの開発者も使っているソフトウェアの 1 つを体験する良い機会でもあると考えている．

[11] `http://e-words.jp/`

Q24. 提出時のソースコードの文字コードは？

Eclipse プロジェクトに対して「プロジェクト用の文字コードの設定」として保存してあれば何でもよいが，UTF-8 を推奨している．

自宅での学習に関する話

Q25. Cygwin を Windows で端末使う場合お勧めの環境は？

Cygwin に Tera Term SSH を組み合わせるのが 1 つの方法である．カレントディレクトリとして使いたいフォルダを右クリックし，コンテキストメニューから “Cygterm Here” で Cygwin 環境を開くことができるので便利である．また，Tera Term SSH 側で入出力の文字コードを指定できるので，様々な文字コードのファイルの出力をコンソール上で読み取りたい場合にも便利である．

Q26. Cygwin 上の javac でファイルを 1 つずつコンパイルするとエラーになるのはなぜ？

たとえば `Main.java` をコンパイルするとき，その中で参照している `BoardView` のようなクラスが「未知のクラス」として扱われてコンパイルエラーの原因となることがある．これは，`javac` コンパイラが「知っている」クラスが，標準ライブラリのクラスと，コンパイル対象のクラスだけになっているためである（Eclipse はプロジェクトに所属する全ファイルを知っているが，`javac` にはそのような情報を得る方法がない）．`javac *.java` のようにファイルを複数指定するか，`-classpath` オプションで `BoardView` などのコンパイル結果のファイルが保存されたディレクトリを参照する必要がある．

Q27. 演習室から git などを使うことはできないのか？

演習室では HTTP 以外のポートは利用制限がかかっていて，SSH などでの外部との通信はできない．Web インタフェース経由でやり取りするものに利用は限られる．

Q28. Eclipse でエクスポートしたものをインポートする方法は？

日本語化した Eclipse で，バージョンも異なるが，解説サイトが存在する [12]．

Q29. 効率よくファイルを自宅・演習室で行き来させるには？

確実に移動させたい場合はプロジェクトのエクスポート/インポートを用いる．プロジェクト情報に変化がない場合は，単にソースファイルだけを持ち運び，上書きコピーすることも可能である．Eclipse で作成しているソースファイルは，

```
workspace/lifegame09Bxxxxx/src/lifegame/*.java
```

という形で保存されているので，メール添付などの方法で移動させればよい．ファイルを上書きした後は，Eclipse 側でプロジェクトを選び「最新の状態に更新（Refresh）」を実行し，ファイルの最新の状態を Eclipse に認識させる必要がある．

[12] `http://java2005.cis.k.hosei.ac.jp/materials/lecture01/eclipse.html`