TUGAS PROJECT

STRUKTUR DATA LINEAR

KELAS E

“ Solitaire (Stack) ”

DISUSUN OLEH :

Paulus Caesario Dito Putra Hartono (205314159) a.

Kurniawan Ronaldi Purnama (205314148) a.

TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS SANATA DHARMA YOGYAKARTA

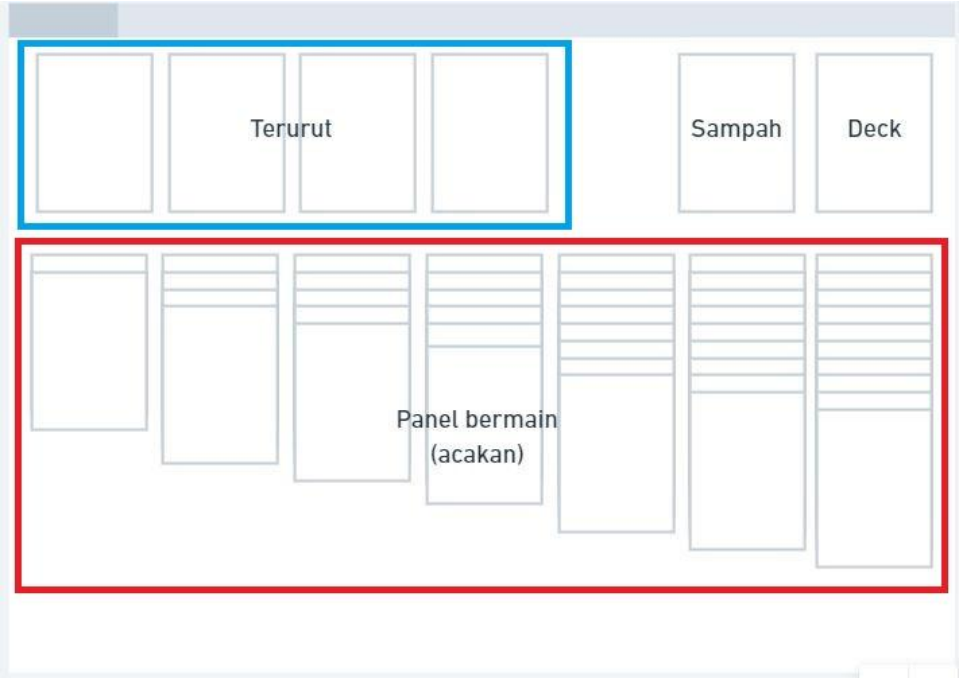
2021

A. Tujuan

Tujuan pembuatan program digunakan untuk memahami konsep Stack. Pada project kali ini kami menerapkan Stack untuk membuat sebuah Game Aplikasi Solitaire dengan aturan permainan yang sudah ditentukan.

1. Alur permainan

a. Pada permainan akan terdapat 4 Jenis tumpukan (stack) yang akan dimainkan dengan nilai 1 - 13, nilai 1 akan merepresentasikan nilai AS, nilai 11 akan merepresentasikan nilai Jack, nilai 12 akan merepresentasikan nilai Queen, dan nilai 13 akan merepresentasikan nilai King.
b. Pada awal permainan akan membuat sebuah tumpukan kartu yang memiliki jumlah total 52 kartu, dan posisi kartu tersebut dalam keadaan acak.
c. Kemudian dari 52 kartu tersebut di-*pop* untuk di-*push* ke dalam tumpukan kartu yang akan dimainkan (panel bermain) sebanyak total kartu 35. Kemudian sisa kartu akan ditaruh (push) ke dalam deck dan sisanya seperti cadangan dan terurut akan memiliki nilai awalan kosong ketika awal bermain.
d. Game akan selesai bila total kartu yang ada di terurut sebanyak (13x4) 52 kartu.berikut adalah kerangka tampilan yang akan digunakan.



2. Kondisi Permainan

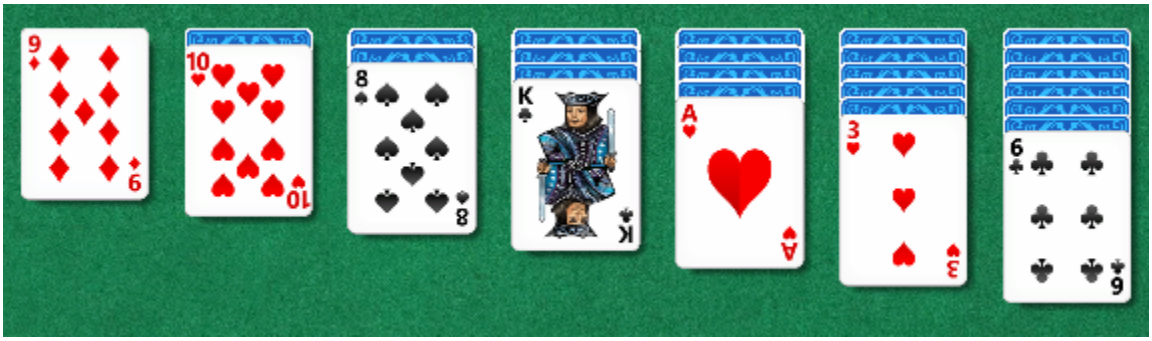
Kondisi Game Berakhir (Menang) :

- Field terurut sudah lengkap (secara terurut), *field* acak sudah habis, *field* deck juga habis.

- Game tidak akan berakhir hingga kondisi tersebut terpenuhi, namun bila pengguna menemukan hal yang tak terduga hingga membuat kondisi dimana kartu tersebut tidak bisa dipindahkan, maka yang harus dilakukan oleh user tersebut harus mengulang permainan karena tidak ada pengecekan bila tidak ada kartu yang bisa dipindahkan.

3. Aturan Bermain

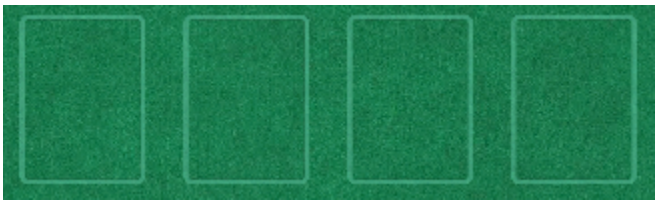
1. Di dalam permainan terdapat 4 field berbeda dengan restriction masing - masing. Berikut penjelasan field:
   - Field Acak



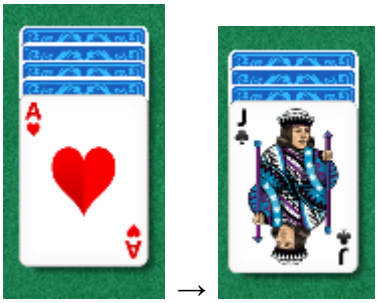
   - Field Deck

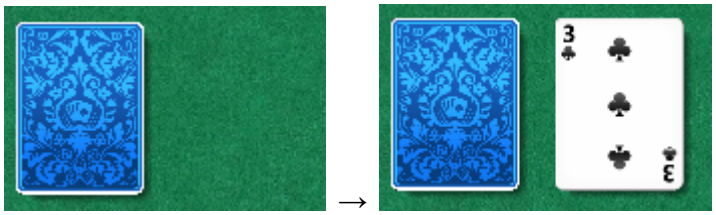
- Field Sampah

- Field Terurut



2. *Restriction* masing - masing field:
   - Acak:
     - ➢ Hanya dapat dipindahkan ke dalam *Field* terurut dan *Field* acak lainnya, jika kondisi kartu dari *field* acak memenuhi untuk dipindahkan ke dalam *field* terurut. Kondisi tersebut jenis kartu harus sama dan nilai yang dipindahkan dengan nilai yang ada di field terurut adalah (urut == acak -1). Contoh bila di field terurut terdapat nilai 9 maka yang dipindahkan untuk di-push ke dalam field terurut nilainya harus 10 karena nilai 10 bila dikurangi 1 nilainya adalah 9 (9 == 10 - 1).
     - ➢ Nilai dari masing - masing acak berbeda - beda. Jika bagian teratas salah satu *Stack* acak dipindahkan ke dalam *field* terurut, maka Data di bawahnya akan terbuka dan dapat digunakan.



- Deck:
  - ➢ Posisi awal deck akan tertutup, dan user tidak mengetahui nilai dibalik kartu paling atas tersebut.
  - ➢ Deck merupakan sisa kartu dari *field* acak, dan bisa digunakan sebagai nilai bantuan bila user kesulitan menemukan langkah berikutnya.
  - ➢ Untuk menggunakan deck, user hanya bisa meng-click field tersebut. Setelah di-click, bagian kartu paling atas akan dipindah ke dalam field sampah.
  - ➢ 
  - ➢ Pengambilan kartu dari field Deck dapat dilakukan berulang - ulang kali sampai kartu yang ada di deck tersebut habis.

- Sampah
  - ➢ Kartu dari sampah hanya bisa dipindahkan ke *field* Terurut.
  - ➢ Jumlah kartu yang dimuat di dalam sampah adalah 0.
  - ➢ Sampah akan terisi bila user menggunakan Deck.
  - ➢ Kartu yang berada di bawah kartu paling atas, tidak dapat digunakan sampai kartu yang paling atas hilang (pindah ke *field* terurut atau *field* acak).
  - ➢ 

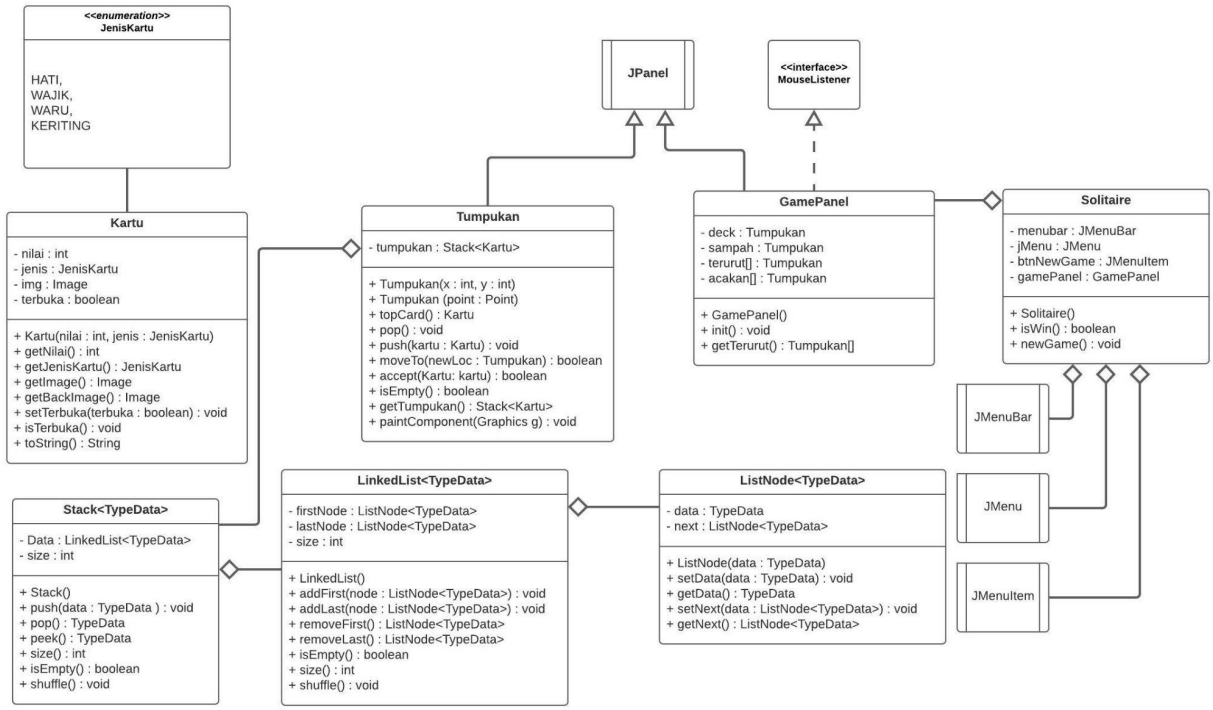
- Terurut
  - ➢ Nilai awal Terurut adalah kosong namun memiliki restriction bahwa kartu yang dimasukan ke dalam field tersebut harus sejenis.
  - ➢ *Field* tersebut bisa menerima kartu dari *field* lain (acak dan deck).
  - ➢ Kartu yang terdapat pada *Field* terurut dapat diambil, dan dipindah ke *Field* Acak.
  - ➢ Masing - masing *stack* di *field* Terurut harus berisi jenis kartu yang sama dan terurut mulai dari AS:1 sampai King:13.

➢ 

Diibaratkan alias *stack* dari kiri ke kanan secara terurut adalah 1, 2, 3, 4. *Stack* 1 tidak dapat diisi dengan kartu jenis lainnya selain Waru atau Sekop (kasus dalam gambar). Hal tersebut berlaku ke *stack - stack* lainnya.

Contoh pengisian *Stack 3* dengan jenis kartu yang lainnya. Kartu bernilai 2 (Wajik) tersebut tidak dapat masuk ke *Stack 3.*

B. Rancangan Struktur Data (Diagram UML)



C. Analisa Method Penting (Penggunaan stack)

Kelas Tumpukan

Kelas Tumpukan meng-*extends* JPanel dan mempunyai attribute Stack dengan tipe Kartu (Stack<Kartu>) yang bernama tumpukan. Constructor dari class tersebut mengandung parameter bernilai (int x, int y) dan Point sebagai lokasi yang akan digunakan pada Component kartu tersebut. Kenapa kelas Tumpukan harus mewarisi kelas JPanel, supaya bisa digunakan untuk mengimplementasikan gambar kartu dengan mudah, serta mengatur posisi tersebut ke dalam gui lebih fleksibel.

Berkaitan dengan struktur data, method yang berhubungan dengan hal tersebut ada:

topCard

```
public Kartu topCard() {
    if (!this.tumpukan.isEmpty()) {
        return this.tumpukan.peek();
    }
    return null;
}
```

→ Method ini digunakan untuk mendapatkan kartu paling atas pada Stack tersebut dengan memanggil method peek(), method peek() memiliki fungsi yang mirip dengan method pop() namun method ini akan mengambil object terakhir tanpa menghilangkan object tersebut dari stack. Method ini nantinya akan digunakan pada method accept sebagai pengecekan kartu yang akan dipindahkan ke tumpukan lainnya.

pop

→ Pop stack pada attribute tumpukan. Menghilangkan objek yang berada di bagian akhir stack.

push

→ Push data pada attribute stack bernama tumpukan.memasukan objek ke dalam stack di posisi akhir.

moveTo

```
public boolean moveTo(Tumpukan newLoc) {
    if (newLoc.accept(this.topCard())) {
        newLoc.push(this.pop());
        if (!this.isEmpty()) {
            this.topCard().setTerbuka(true);
        }
        return true;
    }
    return false;
}
```

→ Tipe data yang akan di-*return* adalah boolean yang digunakan untuk mengetahui kartu berhasil dipindahkan atau tidak. Method ini mempunyai parameter bertipe data Tumpukan yang digunakan untuk memindahkan kartu dari Tumpukan ini ke Tumpukan lainnya (parameter). Di method ini dilakukan validasi untuk mengecek apakah memang kartu tersebut dapat dipindah dari *field* satu ke *field* lainnya. Kenapa diperlukan validasi? Karena digunakan untuk mengecek apakah kartu yang dipindahkan ini sejenis dan apakah

nilai kartu paling atas ini bila dikurangi 1 memiliki nilai yang sama dengan kartu yang berada di tumpukan lain tersebut.

getTumpukan
→ Getter untuk attribute tumpukan.

Class GamePanel - Method init()

Merupakan method yang digunakan untuk inisialisasi *field* - *field* yang tersedia di dalam program tersebut (yang sudah dijelaskan pada halaman sebelumnya). Method ini meng-*generate,* mengacak, serta menempatkan kartu sesuai dengan porsinya masing - masing.

```
39      public void init() {
40          JenisKartu[] jenisKartu = JenisKartu.values();
41          deck = new Tumpukan(DECK_POSITION.x, DECK_POSITION.y);
42          terurut = new Tumpukan[4];
43          sampah = new Tumpukan(DECK_POSITION.x - MARGIN, DECK_POSITION.y);
44          acakan = new Tumpukan[10];
45          for (JenisKartu jenis : jenisKartu) {
46              for (int j = 1; j <= 13; j++) {
47                  deck.push(new Kartu(jenis, j, false));
48              }
49          }
50          deck.getTumpukan().shuffle();
51          for (int i = 0; i < jenisKartu.length; ++i) {
52              terurut[i] = new Tumpukan(20 + MARGIN * i, 20);
53              terurut[i].push(new Kartu(jenisKartu[i], 0, true));
54              add(terurut[i]);
55          }
```

Method tersebut diawali dengan pengambilan nilai enum (baris 40), inisialisasi object deck, terurut, sampah, serta acakan (baris 41-44) dengan porsinya masing - masing. Untuk deck dan sampah langsung menentukan posisi *field* tersebut di dalam panel kelas GamePanel ini. Untuk terurut dan acakan, karena berbentuk array, maka perlu didefinisikan terlebih dahulu panjang dari array tersebut.

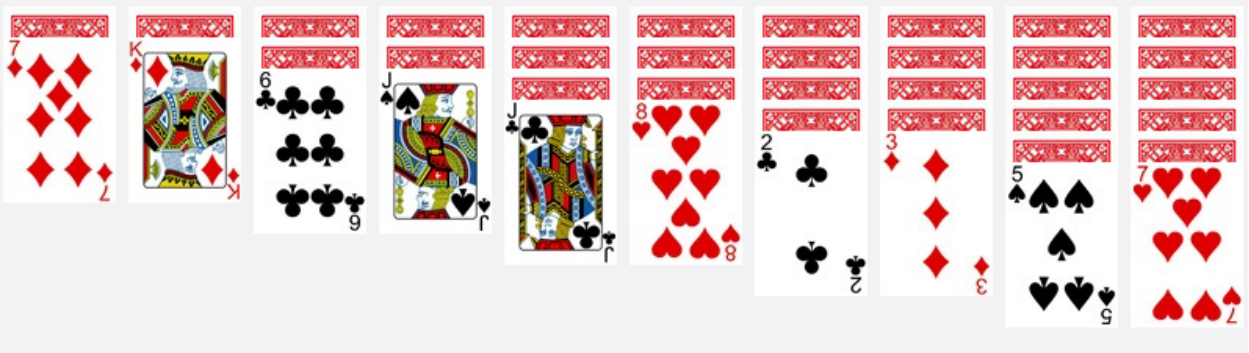
Pada baris 45 - 49, dilakukan perulangan nested untuk setiap jenis kartu sebanyak 13x sebagai isi dari *field* deck. Di dalam perulangan tersebut dilakukan perintah push Kartu sesuai dengan masing - masing jenis serta berawal dari nilai 1 (As) - 13 (King). Setelah perulangan selesai, digunakan method shuffle yang berada pada Class LinkedList untuk me-*randomize* nilai yang terdapat pada deck tersebut.

Dilanjutkan dengan perulangan for pada baris 51- 55 untuk mengisi nilai dari *field* terurut. Untuk setiap jenis kartu dilakukan inisialisasi Point untuk menempatkannya di dalam panel kelas GamePanel ini dan dilakukan push Kartu untuk memberi jumlah kartu Tumpukan, kartu tersebut diambil dari deck, bila perulangan ini selesai maka sisa kartu

dari deck akan tetap disimpan pada deck tersebut. Di akhir perulangan dilakukan add(Component) supaya dapat terlihat pada panel GamePanel ini.

```
56          int sizeAcakan = 1;
57          for (int i = 1; i <= acakan.length; ++i) {
58              acakan[i - 1] = new Tumpukan(ACAKAN_POSITION.x + MARGIN * (i - 1),
59                      ACAKAN_POSITION.y) {
60                  @Override
61                  public void paintComponent(Graphics g) {
62                      super.paintComponent(g);
63                      int yOffset = 0;
64                      if (!this.isEmpty()) {
65                          for (int k = 0; k < this.getTumpukan().size(); k++) {
66                              if (k + 1 == this.getTumpukan().size()) {
67                                  g.drawImage(this.topCard().getImage(), 0,
68                                          yOffset, IMG_WIDTH, IMG_HEIGHT, null);
69                                  yOffset += 20;
70                              } else {
71                                  g.drawImage(this.topCard().getBackImage(), 0,
72                                          yOffset, IMG_WIDTH, IMG_HEIGHT, null);
73                                  yOffset += 20;
74                              }
75                          }
76                          this.setSize(IMG_WIDTH, IMG_HEIGHT + (yOffset - 20));
77                      }
78                  }
79              };
80              if (i % 2 == 1) {
81                  sizeAcakan++;
82              }
83              for (int j = 0; j < sizeAcakan; j++) {
84                  acakan[i - 1].push(deck.pop());
85              }
86              if (!acakan[i - 1].topCard().isTerbuka()) {
87                  acakan[i - 1].topCard().setTerbuka(true);
88              }
89              add(acakan[i - 1]);
90          }
91          add(deck);
92          add(sampah);
93      }
```

Selanjutnya adalah perulangan untuk mengisi nilai dari *field* acakan. Pada masing - masing array/tempat di dalam *field* tersebut, dilakukan inisialisasi dengan Tumpukan acakan. Untuk method Override pada baris 61 - 79 merupakan method yang digunakan untuk menampilkan *field* acakan sebagai berikut.



Dapat dilihat di dalam gambar, terdapat posisi kartu yang terbuka serta posisi kartu yang tertutup. Posisi kartu yang tertampil terbuka akan terletak pada data terakhir dari stack. Sedangkan untuk posisi kartu yang tidak terletak pada data terakhir, gambar dari kartu akan di-*set* tertutup. Hal tersebut dapat terjadi karena adanya percabangan (if-else) pada baris 66 - 74. Posisi kartu dapat menurun seperti di-gambar dikarenakan adanya variabel yOffset yang akan digunakan sebagai sumbu x pada constructor dan nilai tersebut akan bertambah seiring terjadinya perulangan.

```
80                  if (i % 2 == 1) {
81                      sizeAcakan++;
82                  }
83                  for (int j = 0; j < sizeAcakan; j++) {
84                      acakan[i - 1].push(deck.pop());
85                  }
86                  if (!acakan[i - 1].topCard().isTerbuka()) {
87                      acakan[i - 1].topCard().setTerbuka(true);
88                  }
89                  add(acakan[i - 1]);
90              }
91              add(deck);
92              add(sampah);
93          }
```

Dilanjut lagi pada baris 80 - 90. Perintah/*code* pada baris tersebut melanjutkan pengisian *field* acakan yang terdapat di dalam perulangan yang dimulai pada baris 57. Pada perulangan dari baris 57 - 90 digunakan algoritma sehingga kartu yang muncul pada *field* acakan secara urut (dari kiri-kanan) akan tertampil sebanyak 2,2,3,3,4,4,5,5,6,6. Oleh karena itu terdapat variabel pembantu pada baris 56 dan if pada baris 80 - 82 sehingga algoritma tersebut dapat berjalan menghasilkan output tersebut.

Perulangan pada baris 83 - 85 digunakan untuk mengisi masing - masing acakan, sebanyak nilai sizeAcakan. Isi dari perulangan tersebut adalah push pada Tumpukan acakan dengan data dari deck yang di-pop. Jadi data dari deck akan dipindahkan ke dalam acakan secara satu per satu melalui perulangan. Deck masih akan terisi, karena total kartu yang terdapat pada acakan adalah 40 kartu dan total dari Deck adalah 13 x 4.

Pada baris 86 - 88 merupakan percabangan untuk mengecek apakah kartu teratas itu tidak terbuka? Jika iya, maka kartu tersebut akan di-set terbuka, yang berarti nilai dari kartu akan tertampil di dalam GUI.

Terakhir pada baris 89, dilakukan add(Component) untuk masing - masing nilai dalam array acakan. Baris 90 merupakan akhir perulangan.

Untuk baris 91 dan 92, dilakukan add(Component) deck serta sampah juga.

Class GamePanel - Method mousePressed

```
    public void mousePressed(MouseEvent e) {
97      source = e.getComponent().getComponentAt(e.getPoint());
98      if (source instanceof Tumpukan) {
99          setCursor(new Cursor(Cursor.HAND_CURSOR));
100     }
101 }
```

Pada dasarnya method ini akan berhubungan dengan method mouseReleased. Kegunaan dari method ini adalah untuk mengambil attribute source yang dimana merupakan tipe data Component untuk mendapatkan nilai dari Kartu yang akan diambil dan dipindahkan.

Class GamePanel - Method mouseReleased

```
105     public void mouseReleased(MouseEvent e) {
106         target = e.getComponent().getComponentAt(e.getPoint());
107         setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
108         if ((target instanceof Tumpukan && source instanceof Tumpukan)
109                 && (target != deck || target != sampah)) {
110             try {
111                 if (!((Tumpukan) source).isEmpty()) {
112                     for (Tumpukan x : terurut) {
113                         if (x == target) {
114                             ((Tumpukan) source).moveTo(((Tumpukan) target));
115                         }
116                     }
117                     for (Tumpukan x : acakan) {
118                         if (x == target) {
119                             if (x.isEmpty()) {
120                                 x.push(((Tumpukan) source).pop());
121                             } else {
122                                 if (x.topCard().getNilai() - 1
123                                         == ((Tumpukan) source).
124                                                 topCard().getNilai()) {
125                                     x.push(((Tumpukan) source).pop());
126                                 }
127                             }
128                             if (!((Tumpukan) source).isEmpty()) {
129                                 ((Tumpukan) source).topCard().setTerbuka(true);
130                             }
131                         }
132                     }
133                     repaint();
134                 }
135             } catch (Exception ep) {
136             }
137         }
    }
```

Method ini akan berhubungan dengan method mousePressed. Method ini akan tereksekusi jika kita sudah me-*release*/melepas *click* mouse kita. Setelah kita melepas, method akan mencari Component sesuai dengan posisi mouse di-*release*. Pada baris 107 dilakukan validasi untuk memfilter supaya Component yang dipindah, berada pada posisi yang sebenarnya/seharusnya. Baris 110 juga merupakan validasi untuk memastikan bahwa Tumpukan dari source tidak kosong, supaya mengecek bahwa tumpukan dari hasil mousePressed tadi memiliki object pada tumpukan yang dimiliki attribut tersebut.

Dilanjut pada baris 111 - 131 merupakan perulangan untuk memindahkan Component dari source (yang diambil dari mousePressed) kepada target (mouseReleased). Kenapa diperlukan perulangan? Karena tipe data *field* terurut serta acakan adalah array, sehingga perlu dilakukan pengecekan data secara satu per satu untuk memindahkannya. Di dalam 2 perulangan tersebut, berisikan berbagai validasi yang

dibutuhkan untuk memastikan bahwa pemindahan Component dari source ke target bisa berjalan dengan semestinya.

Berada di akhir method, pada baris 132 merupakan method repaint() untuk me-*refresh* tampilan gambar - gambar pada GUI, sehingga data yang sudah dipindah dari tempat satu ke tempat lainnya akan di-*update* dan tampilan pada GUI akan berubah.

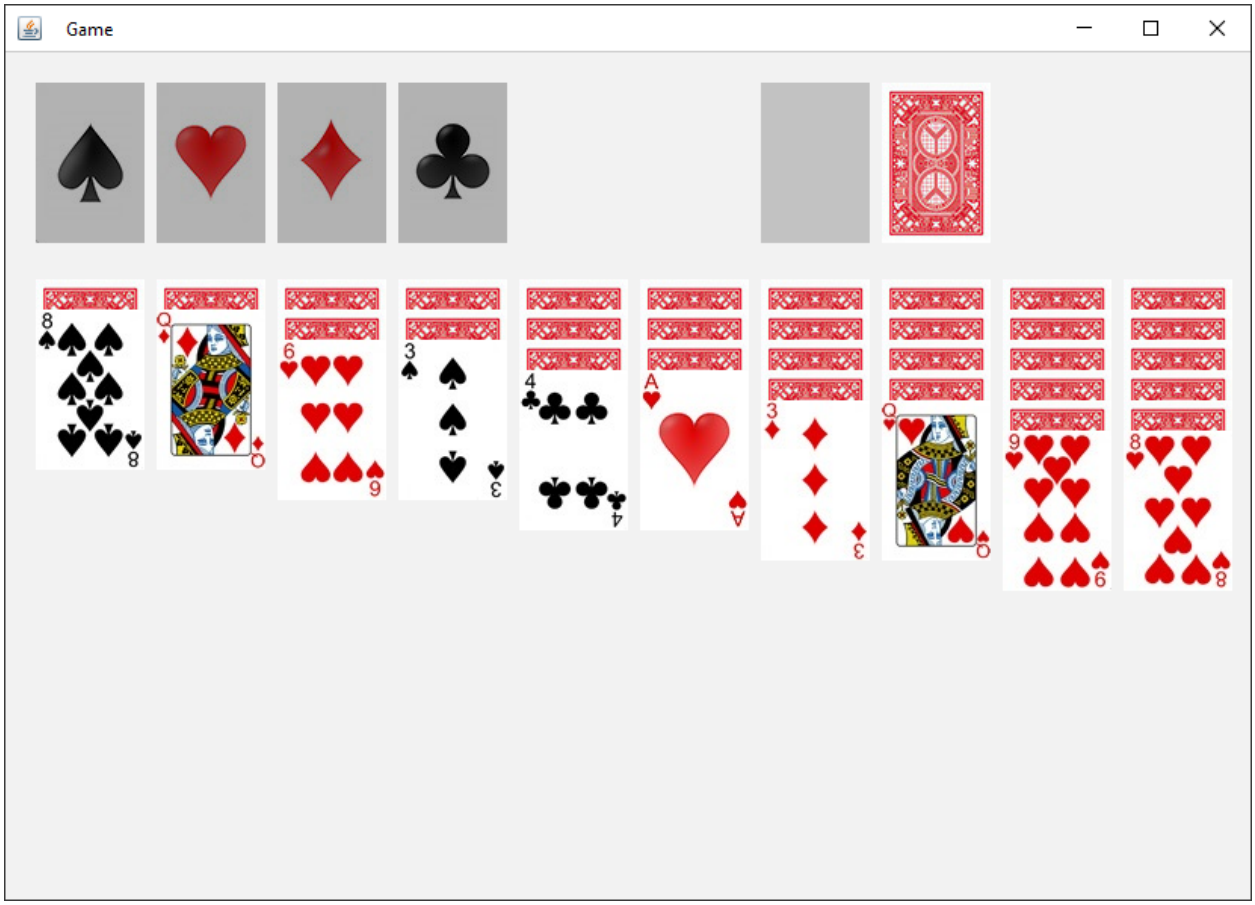
Class GamePanel - Method mouseClicked

```
141     public void mouseClicked(MouseEvent e) {
142         source = e.getComponent().getComponentAt(e.getPoint());
143         if (source == deck) {
144             if (deck.getTumpukan().size() == 0) {
145                 deck.repaint();
146                 deck.setEnabled(false);
147                 deck.addMouseListener(new MouseAdapter() {
148                     @Override
149                     public void mouseClicked(MouseEvent e) {
150                         JOptionPane.showMessageDialog(null, "Kartu Habis");
151                     }
152                 });
153             } else {
154                 if (sampah.getTumpukan().size() != 0) {
155                     sampah.topCard().setTerbuka(false);
156                 }
157                 sampah.push(deck.pop());
158                 sampah.topCard().setTerbuka(true);
159                 sampah.repaint();
160             }
161         }
```

Merupakan method yang dikhususkan untuk *field* deck. Method ini akan tereksekusi jika kita meng-*click* mouse tanpa *drag*. Di dalam method ini terdapat validasi sehingga hanya data dari *field* deck saja yang berubah ketika *user* meng-*click* mouse tersebut. Validasi tersebut berada di baris 142.

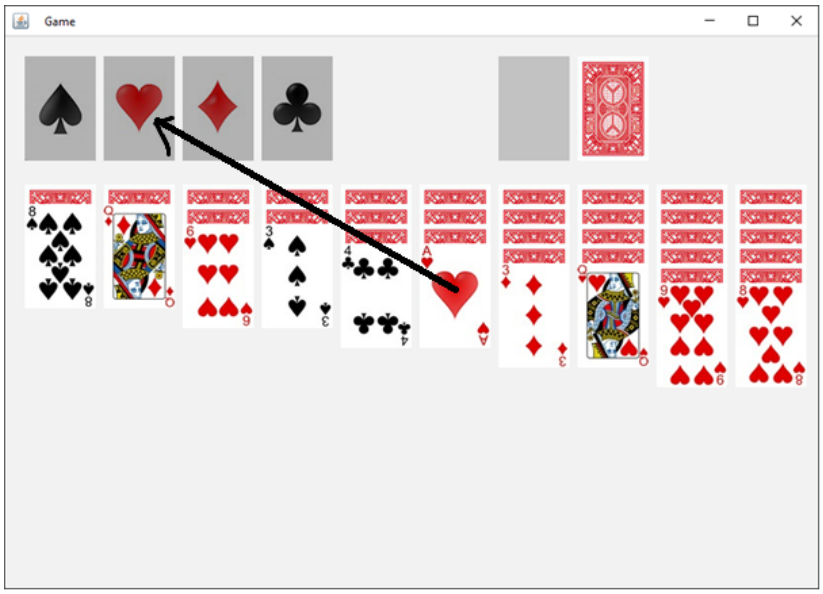
Setelah itu dilakukan lagi validasi lagi untuk mengecek apakah isi dari deck tersebut kosong atau tidak. Jika iya, maka deck sudah tidak bisa dipakai lagi (akan muncul JOptionPane dengan keterangan “Kartu habis”). Jika tidak, maka akan dilakukan push ke *field* sampah dengan data dari *field* deck (pop dari deck). Pada baris 157 digunakan untuk mengubah kartu menjadi tersebut sehingga kartu yang berada di field sampah dapat diakses dan 158 merupakan method untuk me-*refresh* tampilan GUI.

D. Dokumentasi (SS Output & Penjelasan)

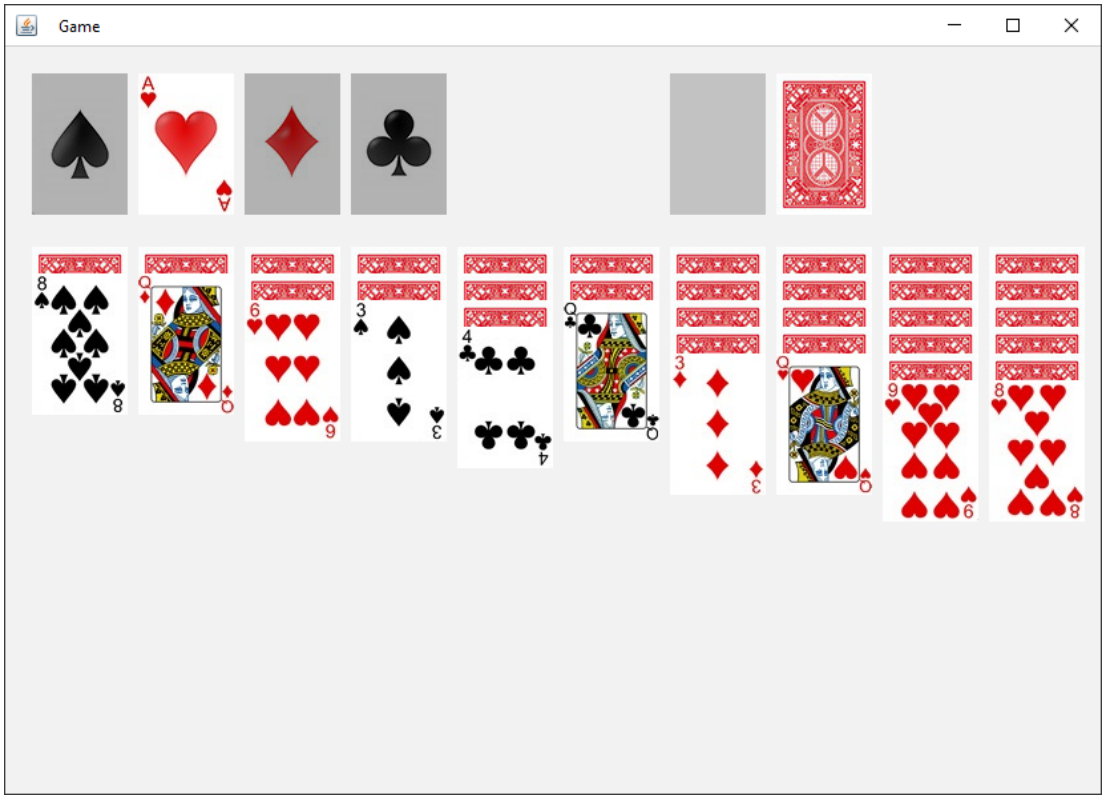


Output pertama program setelah di-*run*.

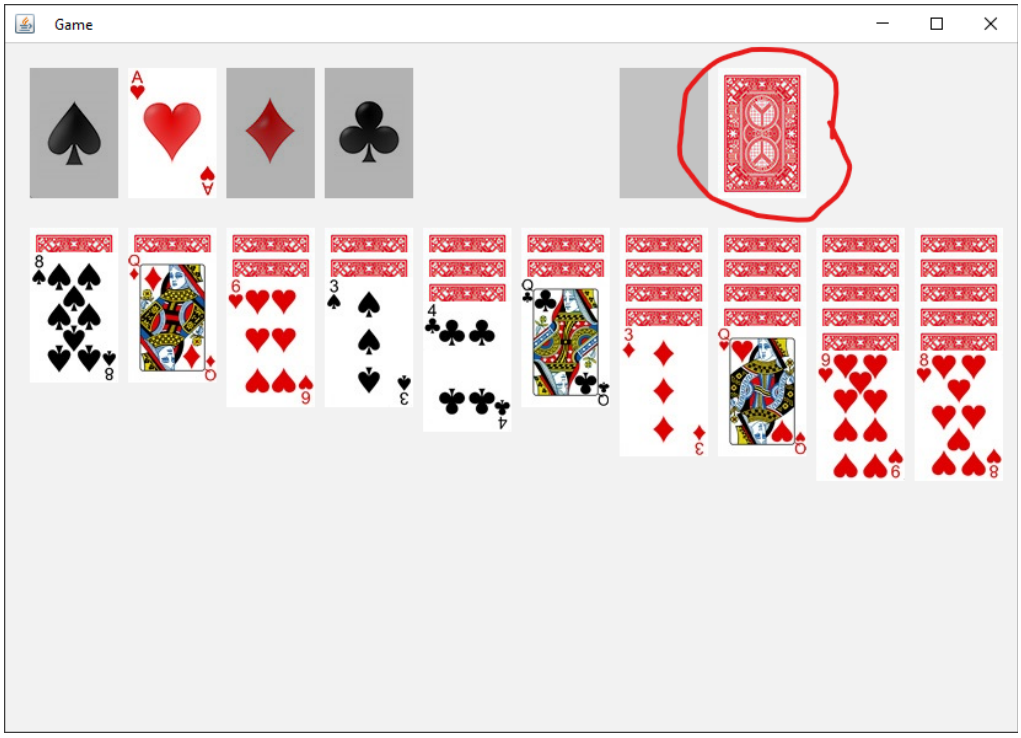
Percobaan pemindahan kartu dari *field* acakan ke *terurut*



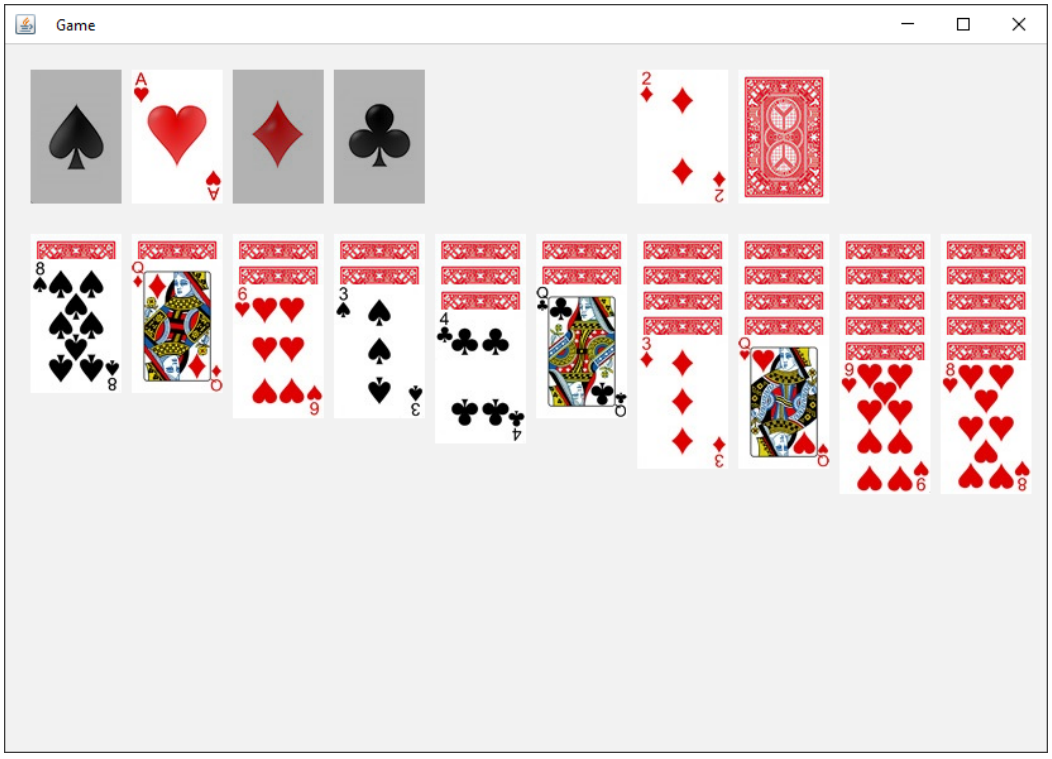
Setelah mouse di *press* dan di *release* maka tampilan akan seperti berikut.



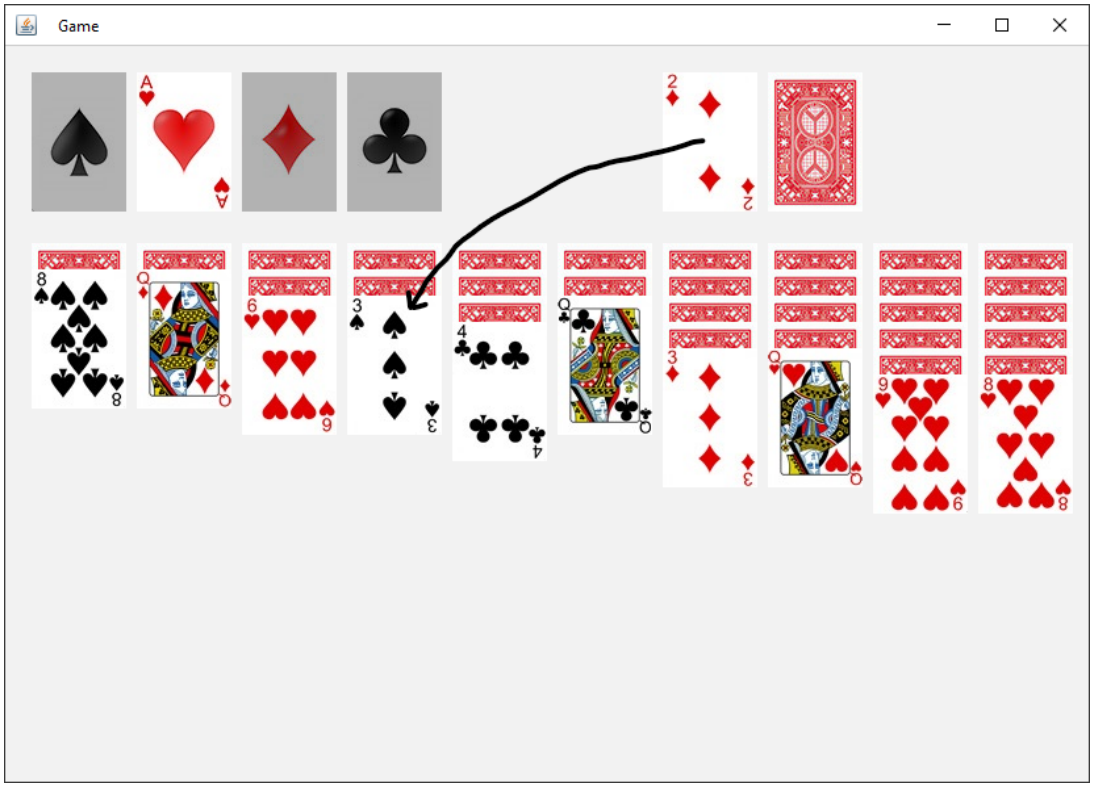
Percobaan penggunaan *field* deck dengan cara meng-*click* kartu pada daerah yang dilingkari



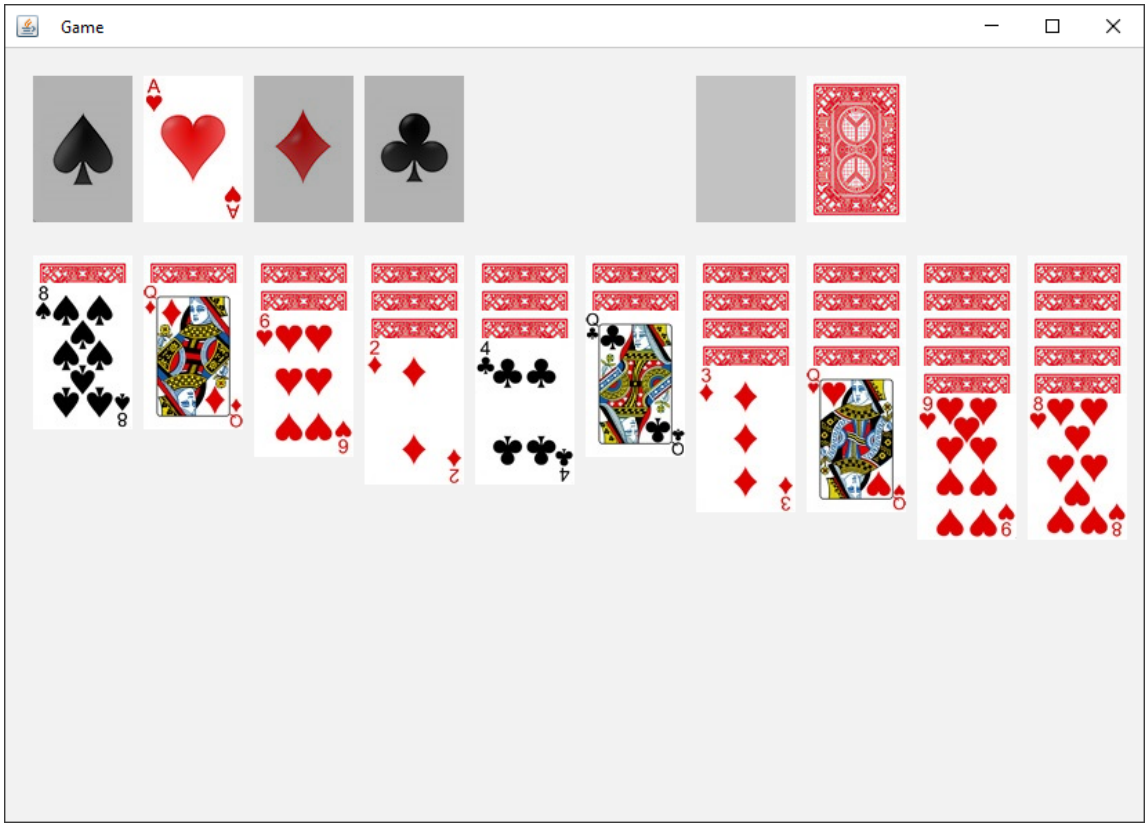
Tampilan setelah di-*click*



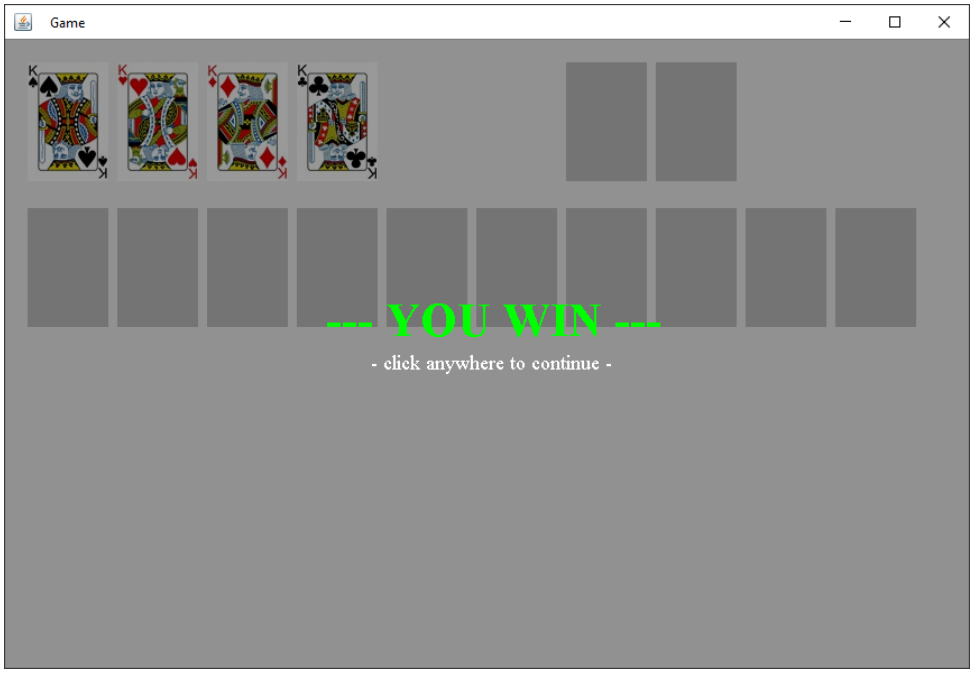
Percobaan pemindahan dari *field* sampah ke *field* acakan.



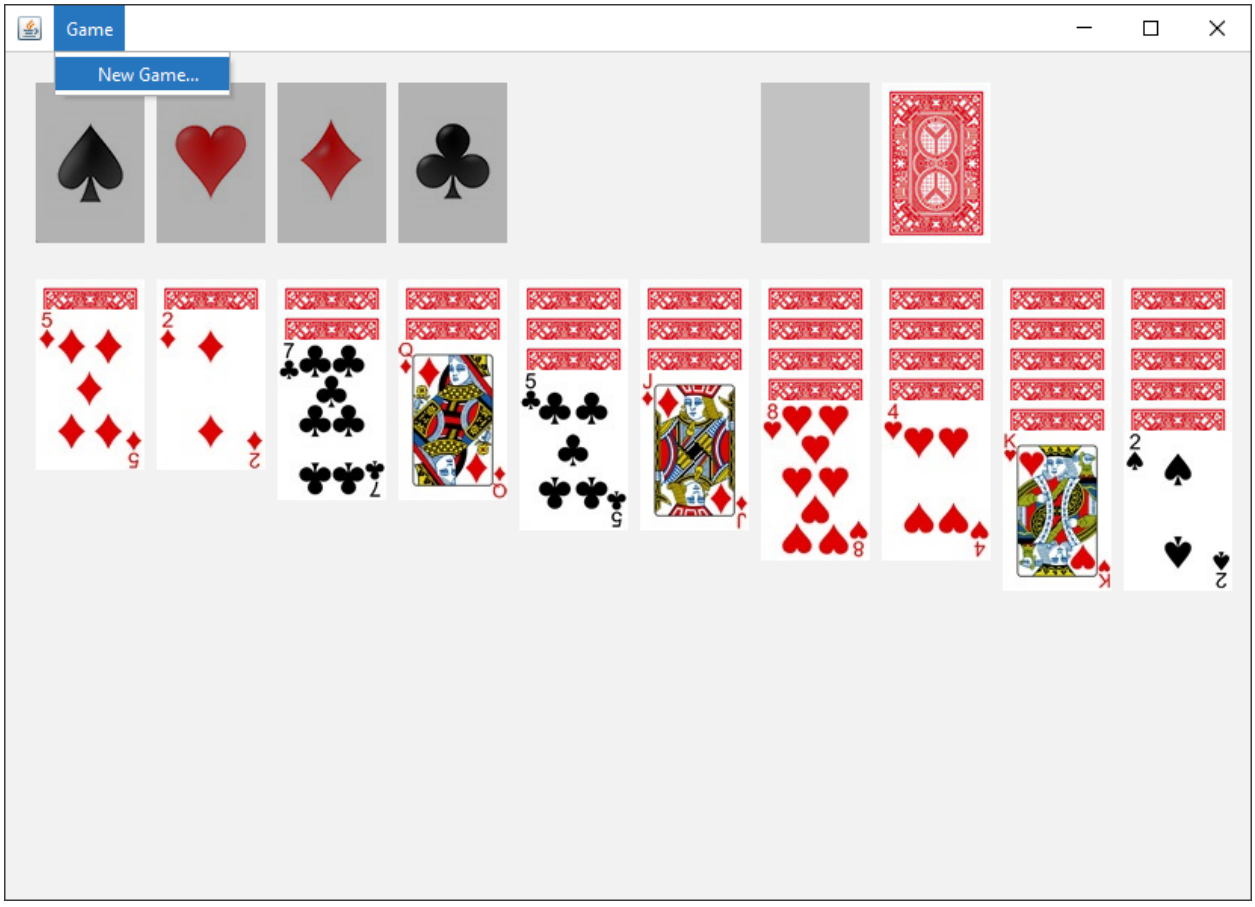
Tampilan setelah di-*drag and drop*.



Tampilan jika semua kartu sudah berada di *field* Terurut dan tidak terdapat sisa di *field* lain, maka akan terjadi output seperti ini dan game akan selesai.



Jika *user* merasa game sudah tidak bisa dimainkan/tidak terdapat gerakan lagi yang bisa dilakukan. Maka user dapat meng-*click* Menu bar “Game” dan meng-*click* “New Game” untuk me-reset kartu dari awal.



Lampiran

- Gambar yang digunakan untuk game solitaire tersebut didapatkan dari link berikut
    - Keseluruhan Kartu
      https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/vector-playing-cards/PNG-cards-1.3.zip
    - Bagian Belakang Kartu
      https://www.google.co.id/imgres?imgurl=https://freepikpsd.com/file/2019/10/playing-card-back-png-3-Transparent-Images-214x300.png&imgrefurl=https://freepikpsd.com/playing-card-back-png-transparent-images/312321/playing-card-back-png-3-transparent-images/&tbnid=CnDsb5pEmkPnGM&vet=1&docid=1za9h5JTXDul2M&w=214&h=300&hl=en-ID&source=sh/x/im
    - Referensi Game
      https://www.microsoft.com/id-id/p/microsoft-solitaire-collection/9wzdncrfhwd2?activetab=pivot:overviewtab