**Nama : Arrick Russell Adinoto**

**NIM : 2540123744**

**Kelas : LA01**

**Dosen : D6665 - REZKI YUNANDA, S.Kom., M.Kom**

***Learning Outcomes:***

**LO 1:** Evaluate Basic refactoring and its application.

**LO 2:** Apply Advanced refactoring and its application.

**I. Studi Kasus (100%)**

Dari *code* berikut:

**import** java**.**awt**.\*; import** java**.**awt**.**geom**.\*;**

**import** java**.**awt**.**image**.**BufferedImage**; import** java**.**awt**.**Toolkit**;**

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

class Position **{** public int x**;** public int y**;**

**}**

class Panel **extends** JPanel **implements** ActionListener **{**

private final int BOARD\_WIDTH**;** private final int BOARD\_HEIGHT**;**

private final int PIXEL**;** private final int MAX\_LENGTH**;** private final int MAX\_POS**;** private final int DELAY **=** 100**;** private final int HEAD **=** 1**;** private final int BODY **=** 2**;** private final int FOOD **=** 3**;** private final Position pos**[];**

private int length**;** private int food\_x**;** private int food\_y**;**

private int LEFT **=** 1**;** private int RIGHT **=** 2**;** private int UP **=** 4**;** private int DOWN **=** 8**;** private int move **=** RIGHT**;**

private boolean alive **= true;**

private boolean moved **= false;**

private Timer timer**;** private Image body**;** private Image food**;** private Image head**;**

private String msg **=** "Game Over"**;**

public Panel**(**int width**,** int height**,** int pixel**) { this.**BOARD\_WIDTH **=** width**;**

**this.**BOARD\_HEIGHT **=** height**; this.**PIXEL **=** pixel**;**

**this.**MAX\_LENGTH **=** BOARD\_WIDTH**\***BOARD\_HEIGHT**; this.**MAX\_POS **= (**BOARD\_WIDTH**/**PIXEL**)-**2**;**

**this.**pos **= new** Position**[**MAX\_LENGTH**]; for(**int i**=**0**;**i**<**pos**.**length**;**i**++)**

pos**[**i**] = new** Position**();**

addKeyListener**(new** TAdapter**());** setBackground**(**Color**.**black**);** setFocusable**(true);**

setPreferredSize**(new** Dimension**(**BOARD\_WIDTH**,** BOARD\_HEIGHT**));** head **=** setupImages**(**HEAD**);**

body **=** setupImages**(**BODY**);** food **=** setupImages**(**FOOD**);** initGame**();**

# }

private Image setupImages**(**int type**) {**

BufferedImage image **= new** BufferedImage**(**PIXEL**,** PIXEL**,** BufferedImage**.**TYPE\_INT\_RGB**);**

Graphics2D g2 **=** image**.**createGraphics**(); switch(**type**) {**

**case** BODY**:**

g2**.**setPaint**(**Color**.**blue**);** g2**.**fillOval**(**0**,** 0**,** PIXEL**,** PIXEL**);**

g2**.**dispose**(); break;**

**case** HEAD**:**

g2**.**setPaint**(**Color**.**red**);** g2**.**fillOval**(**0**,** 0**,** PIXEL**,** PIXEL**);**

g2**.**dispose**(); break;**

**case** FOOD**:**

g2**.**setPaint**(**Color**.**green**);** g2**.**fillOval**(**2**,** 2**,** PIXEL**-**2**,** PIXEL**-**2**);**

g2**.**setPaint**(new** Color**(**102**,**51**,**0**));** int **[]** x **= {**1**,** PIXEL**/**2**,** PIXEL**/**2**};** int **[]** y **= {**1**,** PIXEL**/**2**,** 1**};**

g2**.**fillPolygon**(**x**,**y**,**3**);** g2**.**dispose**();**

# break;

**}**

**return** image**;**

# }

private void initGame**() {**

length **=** 3**;**

**for (**int i**=**0**;** i**<**length**;** i**++) {** pos**[**i**].**x **=** PIXEL**\***5 **-** i **\*** 10**;** pos**[**i**].**y **=** PIXEL**\***5**;**

# }

randomFood**();**

timer **= new** Timer**(**DELAY**, this);**

timer**.**start**();**

# }

@Override

public void paintComponent**(**Graphics g**) { super.**paintComponent**(**g**);**

**if (**alive**) {**

g**.**drawImage**(**food**,** food\_x**,** food\_y**, this);** g**.**drawImage**(**head**,** pos**[**0**].**x**,** pos**[**0**].**y**, this); for (**int i **=** 1**;** i **<** length**;** i**++)**

g**.**drawImage**(**body**,** pos**[**i**].**x**,** pos**[**i**].**y**, this);**

Toolkit**.**getDefaultToolkit**().**sync**();**

# } else {

gameOver**(**g**);**

# }

**}**

private void gameOver**(**Graphics g**) {**

Font small **= new** Font**(**"Helvetica"**,** Font**.**BOLD**,** 14**);** FontMetrics metr **=** getFontMetrics**(**small**);**

g**.**setColor**(**Color**.**white**);** g**.**setFont**(**small**);**

g**.**drawString**(**msg**, (**BOARD\_WIDTH **-** metr**.**stringWidth**(**msg**)) /** 2**,** BOARD\_HEIGHT **/** 2**);**

# }

private void randomFood**() {**

int r **= (**int**) (**Math**.**random**() \*** MAX\_POS**)+**1**;** food\_x **= ((**r **\*** PIXEL**));**

r **= (**int**) (**Math**.**random**() \*** MAX\_POS**)+**1**;** food\_y **= ((**r **\*** PIXEL**));**

# }

@Override

public void actionPerformed**(**ActionEvent e**) { if (**alive**) {**

// check food

**if ((**pos**[**0**].**x **==** food\_x**) && (**pos**[**0**].**y **==** food\_y**)) {**

length**++;** randomFood**();**

int d **=** DELAY**-**length**; if(**d **<** 40**)** d **=** 40**;** timer**.**setDelay**(**d**);**

# }

// checkCollision

**for (**int i**=**length**;** i**>**0**;** i**--) {**

**if (**pos**[**0**].**x **==** pos**[**i**].**x **&&** pos**[**0**].**y **==** pos**[**i**].**y**) {**

alive **= false;**

# }

**}**

**if (**pos**[**0**].**y **>=** BOARD\_HEIGHT**)**

alive **= false; if (**pos**[**0**].**y **<** 0**)**

alive **= false;**

**if (**pos**[**0**].**x **>=** BOARD\_WIDTH**)**

alive **= false; if (**pos**[**0**].**x **<** 0**)**

alive **= false;**

**if (!**alive**)**

timer**.**stop**();**

//move

**for (**int i**=**length**;** i**>**0**;** i**--) {**

pos**[**i**].**x **=** pos**[**i**-**1**].**x**;**

pos**[**i**].**y **=** pos**[**i**-**1**].**y**;**

# }

**if (**move **==** LEFT**)** pos**[**0**].**x **-=** PIXEL**;**

**if (**move **==** RIGHT**)** pos**[**0**].**x **+=** PIXEL**;**

**if (**move **==** UP**)** pos**[**0**].**y **-=** PIXEL**;**

**if (**move **==** DOWN**)** pos**[**0**].**y **+=** PIXEL**;**

moved**=true;** repaint**();**

# }

**}**

private class TAdapter **extends** KeyAdapter **{**

@Override

public void keyPressed**(**KeyEvent e**) { if(!**moved**) return;**

int key **=** e**.**getKeyCode**(); switch(**key**) {**

**case** KeyEvent**.**VK\_LEFT**:**

**if(**move**!=**RIGHT**)** move **=** LEFT**; break;**

**case** KeyEvent**.**VK\_RIGHT**:**

**if(**move**!=**LEFT**)** move **=** RIGHT**; break; case** KeyEvent**.**VK\_UP**:**

**if(**move**!=**DOWN**)** move **=** UP**; break; case** KeyEvent**.**VK\_DOWN**:**

**if(**move**!=**UP**)** move **=** DOWN**; break;**

**}**

moved **= false;**

**}**

**}**

**}**

public class Snake **extends** JFrame **{**

public Snake**() {**

add**(new** Panel**(**600**,** 600**,** 20**));** setResizable**(false);**

pack**();**

setTitle**(**"Snake Game"**);** setLocationRelativeTo**(null);** setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

**}**

public static void main**(**String**[]** args**) {**

EventQueue**.**invokeLater**(() -> {** JFrame ex **= new** Snake**();** ex**.**setVisible**(true);**

**});**

**}**

**}**

**[LO 1 & LO 2, 100 poin]** Cari **5** jenis ***code smell*** yang berbeda dan lakukan ***refactoring*** untuk *code smell*- nya (masing-masing *code smell* dan *refactoring* bernilai **20 poin**). Tuliskan jawaban Anda dengan format berikut dalam 1 *file* doc:

1.

|  |
| --- |
| Dispensables : Data Class |
| Code ini memiliki smell dispensables dengan spesifik Smell yaitu Data Class. Data class merupakan sebuah code smell dimana class hanya berisikan sebuah field yang bisa diakses secara langsung oleh class lain. Sepintas memang tidak terlihat bermasalah, namun dalam pengembangan kode, jika terus menerus field tidak dienkapsulasi, maka field dapat diubah oleh class lain menjadi sebuah value yang tidak masuk akal ataupun tanpa persyaratan dari class position.  class Position {      public int x;      public int y;  } |
| Untuk menyelesaikan smell ini, saya melakukan refactor menggunakan encapsulate field sehingga field dalam class Position tidak dapat diakses secara langsung oleh class lain. Meskipun belum terdapat method-method lain, setidaknya class-class lain sudah terbiasa menggunakan setter dan getter dari class Position sehingga nanti tidak perlu melakukan perubahan secara besar-besaran jika class Position dikembangkan.  class Position {      private int x;      private int y;      public int getX(){          return this.x;      }      public int getY(){          return this.y;      }      public void setX(int x){          this.x=x;      }      public void setY(int y){          this.y=y;      }  } |
|  |

2.

|  |
| --- |
| Dispensables : Dead Code |
| Dalam code ini, import java.awt.geom.\*; tidaak pernah digunakan dalam code. Maka dari itu, import ini termasuk ke dalam Dead Code. Dead Code adalah sebuah bagian dalam code dimana bagian tersebut tidak pernah digunakan di dalam keseluruhan kode. Hal ini dapat membahayakan karena dapat menyebabkan ambiguitas atau kebingunan pada saat code ingin dipelajari ataupun dikembangkan oleh developer yang berbeda.  import java.awt.\*;  import java.awt.geom.\*;  import java.awt.image.BufferedImage;  import java.awt.Toolkit;  import java.awt.event.\*;  import javax.swing.\*; |
| Untuk menyelesaikan ini maka solusi dari saya adalah menghapus bagian code yang tidak digunakan yaitu pada bagian ini adalah import java.awt.geom\*; Dengan ini, maka kode dapat menghilangkan ambiguitasnya dan lebih mudah dipelajari oleh developer karena setiap baris memiliki makna.  import java.awt.\*;  import java.awt.image.BufferedImage;  import java.awt.Toolkit;  import java.awt.event.\*;  import javax.swing.\*; |

3.

|  |
| --- |
| Bloaters : Long Method |
| Pada bagian ini terdapat smell long method dimana method ini sangat Panjang berjumlah sekitar 46 baris, dimana code cenderung susah dipahami dan pengembangan code ke depannya akan lebih sulit untuk dilakukan.  public void actionPerformed(ActionEvent e) {      if (alive) {          // check food          if ((pos[0].x == food\_x) && (pos[0].y == food\_y)) {              length++;              randomFood();              int d = DELAY-length;              if(d < 40) d = 40;              timer.setDelay(d);          }          // checkCollision          for (int i=length; i>0; i--) {              if (pos[0].x == pos[i].x && pos[0].y == pos[i].y) {                  alive = false;              }          }          if (pos[0].y >= BOARD\_HEIGHT)              alive = false;          if (pos[0].y < 0)              alive = false;          if (pos[0].x >= BOARD\_WIDTH)              alive = false;          if (pos[0].x < 0)              alive = false;            if (!alive)              timer.stop();            //move          for (int i=length; i>0; i--) {              pos[i].x = pos[i-1].x;              pos[i].y = pos[i-1].y;          }          if (move == LEFT)              pos[0].x -= PIXEL;          if (move == RIGHT)              pos[0].x += PIXEL;          if (move == UP)              pos[0].y -= PIXEL;          if (move == DOWN)              pos[0].y += PIXEL;          moved=true;          repaint();      }  } |
| Maka dari itu, solusinya adalah dengan menggunakan extract method sehingga code yang ada dipecah2 lagi menjadi method-method yang lebih kecil sehingga code akan lebih mudah untuk dipahami dan arah ekseskusi code pun menjadi lebih jelas. Selain itu, pendeteksian error juga menjadi lebih mudah.      @Override      public void actionPerformed(ActionEvent e) {          if (alive) {              checkFood();              checkCollision();              action\_move();          }      }      public void checkFood(){          if ((pos[0].x == food\_x) && (pos[0].y == food\_y)) {              length++;              randomFood();              int d = DELAY-length;              if(d < 40) d = 40;              timer.setDelay(d);          }      }      public void checkCollision(){          for (int i=length; i>0; i--) {              if (pos[0].x == pos[i].x && pos[0].y == pos[i].y) {                  alive = false;              }          }          if (pos[0].y >= BOARD\_HEIGHT)alive = false;          if (pos[0].y < 0)alive = false;          if (pos[0].x >= BOARD\_WIDTH)alive = false;          if (pos[0].x < 0)alive = false;          if (!alive)timer.stop();      }      public void action\_move(){          for (int i=length; i>0; i--) {              pos[i].x = pos[i-1].x;              pos[i].y = pos[i-1].y;          }          if (move == LEFT)pos[0].x -= PIXEL;          if (move == RIGHT)pos[0].x += PIXEL;          if (move == UP)pos[0].y -= PIXEL;          if (move == DOWN)pos[0].y += PIXEL;          moved=true;          repaint();      } |

4.

|  |
| --- |
| OO Abuser : Switch Statement |
| Pada bagian ini, terdapat object oriented abuser yaitu Switch Statement, dimana code akan ditampilkan dalam bentuk switch statement yang terkesan abusive yaitu switch statement yang berbentuk kotor. Sehingga akan sulit untuk dikembangkan dalam waktu jangka Panjang.   private Image setupImages(int type) {      BufferedImage image = new BufferedImage(PIXEL, PIXEL, BufferedImage.TYPE\_INT\_RGB);      Graphics2D g2 = image.createGraphics();      switch(type) {          case BODY:                  g2.setPaint(Color.blue);                  g2.fillOval(0, 0, PIXEL, PIXEL);                  g2.dispose();              break;          case HEAD:                  g2.setPaint(Color.red);                  g2.fillOval(0, 0, PIXEL, PIXEL);                  g2.dispose();              break;          case FOOD:                  g2.setPaint(Color.green);                  g2.fillOval(2, 2, PIXEL-2, PIXEL-2);                  g2.setPaint(new Color(102,51,0));                  int [] x = {1, PIXEL/2, PIXEL/2};                  int [] y = {1, PIXEL/2, 1};                  g2.fillPolygon(x,y,3);                  g2.dispose();              break;      }      return image;  } |
| Code akan direfactor menggunakan Replace Type Code with Subclasses. Dengan itu code akan lebih mudah untuk dipahami, dibaca serta dikembangkan oleh developer lainnya. Dikarenakan switch code sudah lebih terpecah-pecah sehingga Ketika membaca switch statement, developer akan langsung memahami arah jalan kode secara intuitif.  class ImageSetUp{      BufferedImage image;      Graphics2D g2;      public ImageSetUp(int PIXEL, Color color,int ovalX, int ovalY, int minusPixel){          image = = new BufferedImage(PIXEL, PIXEL, BufferedImage.TYPE\_INT\_RGB);          g2.image.createGraphics();          g2.setPaint(color);          g2.fillOval(ovalX, ovalY, PIXEL-minusPixel, PIXEL-minusPixel);      }  }  class Body extends ImageSetUp{      public Body(int PIXEL){          super(PIXEL, Color.blue,0,0,0);          g2.dispose();      }  }  class Head extends ImageSetUp{      public Head(){          super(PIXEL, Color.red,0,0,0);          g2.dispose();      }  }  class Food extends ImageSetUp{      public Food(){          int [] x = {1, PIXEL/2, PIXEL/2};          int [] y = {1, PIXEL/2, 1};          g2.setPaint(new Color(102,51,0));          g2.fillPolygon(x,y,3);          g2.dispose();      }  }  Code Switch Statement pada Class Panel:      private Image setupImages(int type) {          switch(type) {              case BODY:                      return new Body(this.PIXEL).image;                  break;              case HEAD:                      return new Head(this.PIXEL).image;                  break;              case FOOD:                      return new Food(this.PIXEL).image;                  break;          }      } |

5.

|  |
| --- |
| Bloaters : Primitive Obsession |
| Pada bagian ini, merupakan smell code untuk primitive obsession, dimana code membuat integer untuk food\_x dan food\_y. Padahal food\_x dan food\_y dapat digabungkan menjadi 1 variabel yaitu food yang merupakan object dari class Position.      private int food\_x;      private int food\_y; |
| Untuk memperbaiki permasalahan ini, solusinya adalah dengan Replace Data Value with Object sehingga dapat membuat kode menjadi lebih fleksibel serta lebih mudah untuk ke diorganisir ke depannya.  private Position food; |

-- Selamat Mengerjakan --