



# JAVA APPLETS (2)



# Tiga Prinsip OOP

## Encapsulation

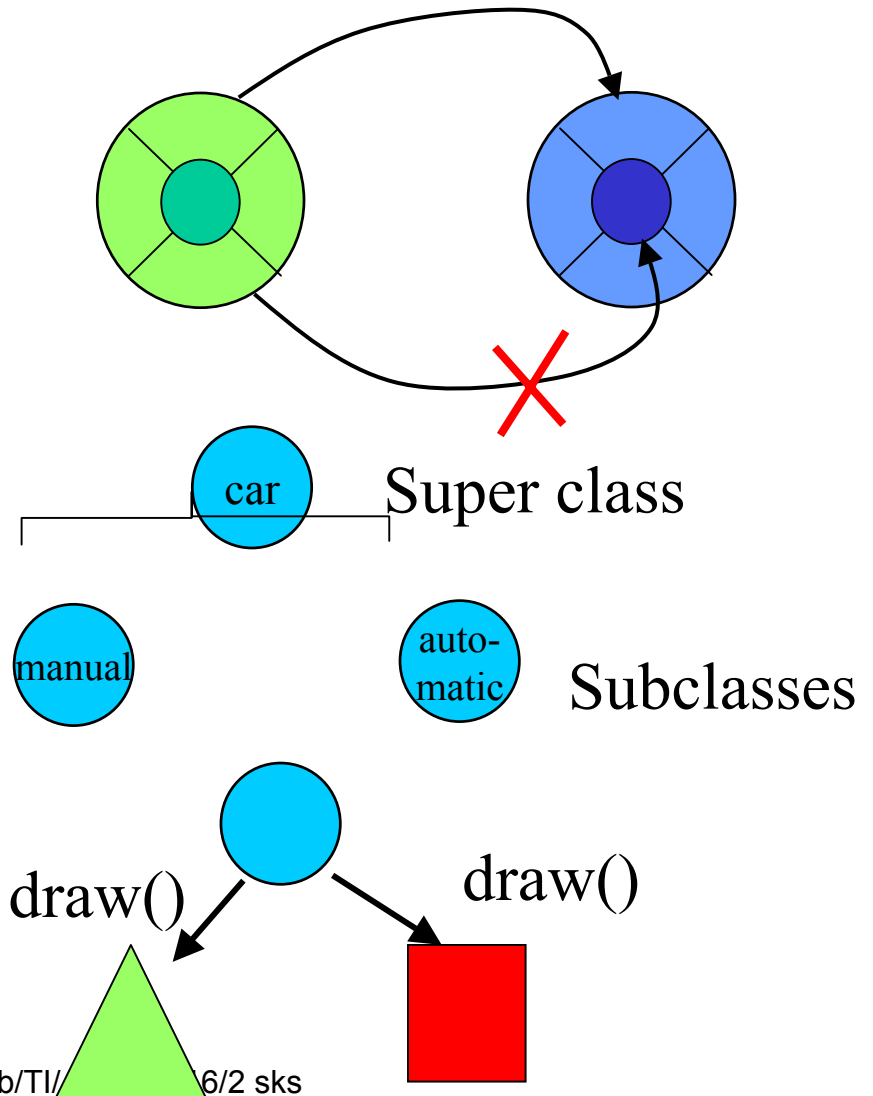
- Object menyembunyikan fungsi mereka (**methods**) dan data (**instance variables**)

## Inheritance

- Setiap **subclass** mewarisi seluruh variable dari **superclass-nya**

## Polymorphism

- Interface sama menghubungkan tipe data berbeda





# Class dan Method Sederhana

```
Class Fruit {  
    int grams;  
    int cals_per_gram;  
  
    int total_calories() {  
        return(grams*cals_per_gram);  
    }  
}
```



# Methods

- Suatu method adalah suatu urutan code yang diberi nama yang dapat dilibatkan oleh Java code lain
- Suatu metoda mengambil beberapa parameter, melaksanakan beberapa perhitungan dan kemudian secara bebas mengembalikan nilai ( atau obyek).
- Methods dapat digunakan sebagai bagian dari statement expression.

```
public float convertCelsius(float tempC) {  
    return( ((tempC * 9.0f) / 5.0f) + 32.0 );  
}
```



# Method Signatures

- Sebuah method signature menentukan :
  - Nama dari method.
  - Type dan nama dari setiap parameter.
  - Type dari value (atau object) yang dikembalikan oleh method.
  - Berbagai macam method modifiers.
  - *modifiers type name ( parameter list ) [throws exceptions ]*

```
public float convertCelsius (float tCelsius ) {}
```

```
public boolean setUserInfo ( int i, int j, String name ) throws  
    IndexOutOfBoundsException {}
```



# Public/private

- Methods/data dapat dideklasikan **public** atau **private** yang artinya method/data tersebut dapat atau tidak dapat diakses oleh code pada class lain ...
- Good practice:
  - keep data private
  - keep most methods private
- Interface yang didefinisikan dengan baik antar class – menolong menghilangkan error

# Using objects

Code pada sebuah class akan membuat sebuah instance dari class lain ...

```
Fruit plum=new Fruit();  
int cal;  
cal = plum.total_calories();
```

**Dot operator** membolehkan kita untuk mengakses (public) data/methods dalam Fruit class



# Constructors

- The line

```
plum = new Fruit();
```

- invokes sebuah constructor method dimana kita dapat men-set initial data dari sebuah object
- Kita dapat memilih beberapa type yg berbeda dari constructor dgn argument lists yg berbeda  
eg Fruit(), Fruit(a) ...

# Overloading

- Dapat memiliki beberapa versi dari sebuah method dalam class dengan tipe/jumlah arguments yang berbeda

```
Fruit() {grams=50;}
```

```
Fruit(a,b) { grams=a; cal_per_gram=b;}
```

- Dengan memperhatikan pada argument, Java memutuskan versi mana yang digunakan



# Java Development Kit

- javac - The Java Compiler
- java - The Java Interpreter
- jdb - The Java Debugger
- appletviewer - Tool to run the applets
  
- javap - to print the Java bytecodes
- javaprof - Java profiler
- javadoc - documentation generator
- javah - creates C header files



# Java vs. C++

## Sintaks Java meminjam dari C++ (dan C)

- primitive types : sama seperti C++, tetapi sizes  
byte (8 bits)    char (16 bits)    short (16 bits)    int (32 bits)  
long (64 bits)    float (32 bits)    double (64 bits)    boolean
- variables, assignments, arithmetic & relational operators : sama seperti C++
- control structures : sama seperti C++, kecuali goto
- Functions : mirip dengan C++, tetapi harus class & harus ditentukan public/private

## in Java, every variable & method belongs to a class

- Seperti di C++, dengan default setiap object mempunyai salinan data fields sendiri sehingga dikenal sebagai *instance variables*
- Seperti di C++, sebuah variables dideklarasikan static bersama dengan seluruh class objects sehingga dikenal sebagai *class variables*
- Hal yang sama, terjadi pada sebuah static method (*class method*)  
Hanya dapat dioperasikan pada class variables, diakses dari class itu sendiri

```
class Math
{ public static final double PI = 3.14159;           // access as Math.PI
  public static double sqrt(double num) { . . . }    // access as in Math.sqrt(9.0)
  . . . }
```



# Primitive vs. Reference Types

primitive types are handled exactly as in C++

- space untuk sebuah primitive object secara implisit dialokasikan  
→ variable mengacu pada actual data (disimpan pada stack)

reference types (classes) are handled differently

- space untuk sebuah reference object secara eksplisit dialokasikan menggunakan `new`  
→ variable mengacu pada sebuah pointer ke data (dimana disimpan pada heap)

*Note: tidak seperti C++, programmer tidak bertanggung jawab untuk menghapus dynamic objects  
JVM melaksanakan automatic garbage collection untuk mereklamasi memory yang tidak digunakan*

Java only provides by-value parameter passing

- Tetapi mengacu pada object yang diimplementasikan sebagai pointers ke dynamic memory
- Menghasilkan behavior mimics by-reference

```
public void Init(int[] nums)
{
    for (int i = 0; i < nums.length; i++) {
        nums[i] = 0;
    }
}
```

---

```
int nums[] = new int[10];
Init(nums);
```



# Java Libraries

- **String class** (secara otomatis di-load dari `java.lang`)

```
int length()
char charAt(index)
int indexOf(substring)
String substring(start, end)
String toUpperCase()
boolean equals(Object)
...
```

```
String str = "foo"
String str = new String("foo");
```

- **Array class** (secara otomatis di-load dari `java.lang`)

```
int length           instance variable
Type [](index)      operator
String toString()
...
```

```
int[] nums = {1,2,3,4,5};
int[] nums = new int[10];
```

- **Java provides extensive libraries of data structures & algorithms**

```
java.util →      Date           Random           Timer
                  ArrayList        LinkedList       Stack
                  TreeSet           HashSet
                  TreeMap          HashMap
```



# Applet Behavior

- recall
  - `init` method dipanggil saat applet diload pertama
  - Berguna untuk initializing variables & objects
  - `paint` method dipanggil segera setelah `init`, dan kapanpun applet perlu menggambar ( contoh : setelah window resized )
- when `paint` is called, it is given the default Graphics object
  - Graphics methods termasuk :


```
void drawString(String msg, int x, int y)
```

```
void setColor(Color color)
```

`Color` class is predefined, constants include:

```
Color.red, Color.blue, Color.black, ...
```

# Hello Again



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

/**
 * This class displays lots of "Hello world!"s on the applet
 * window.
 */
public class HelloWorld2 extends Applet
{
    private static final int NUM_WORDS=100;
    private static final Color[] colors =
    {Color.black,Color.red,Color.blue,Color.green,
    Color.yellow};
    private static Random randy;

    private int RandomInRange(int low, int high)
    {
        return (Math.abs(randy.nextInt()) %
        (high-low+1)) + low;
    }

    public void init()
    {
        randy = new Random();
    }

    public void paint(Graphics g)
    {
        for (int i = 0; i < NUM_WORDS; i++) {
            int x = RandomInRange(1, 140);
            int y = RandomInRange(10, 200);
            g.setColor(colors[RandomInRange(0, 4)]);
            g.drawString("Hello world!", x, y);
        }
    }
}
```

store colors in an array

- pick random index and change color using setColor

Random class provides methods for generating random values

override init method to allocate & initialize (similar to a constructor)

```
<applet code="HelloWorld2.class" height=200 width=200>
You must use a Java-enabled browser to view this applet.
</applet>
```

Pemrograman Web/TI/ AK045216/2 sks

[view page](#)



# Parameters & Applet Dimensions

recall:

- Dapat menentukan parameters di HTML document menggunakan `<PARAM>` tags
- Akses nilai parameter values (berdasar pada nama) menggunakan `getParameter` method

can also access the dimensions of an applet using a Dimension object

```
Dimension dim = getSize(); // stores applet dimensions
```

dapat mengakses applet height melalui `dim.height`

dapat mengakses applet width melalui `dim.width`

# Adaptive Hello



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

/*
 * This class displays lots of "Hello world!"s on the applet window.
 */

public class HelloWorld3 extends Applet
{
    private static final Color[] colors =
        {Color.black,Color.red,Color.blue,Color.green,Color.yellow};
    private static Random randy;
    private Dimension dim;
    private int numReps;

    private int RandomInRange(int low, int high)
    {
        return (Math.abs(randy.nextInt()) % (high-low+1)) + low;
    }

    public void init()
    {
        randy = new Random();
        dim = getSize();
        numReps = Integer.parseInt(getParameter("reps"));
    }

    public void paint(Graphics g)
    {
        for (int i = 0; i < numReps; i++) {
            int x = RandomInRange(1, dim.width-60);
            int y = RandomInRange(10, dim.height);
            g.setColor(colors[RandomInRange(0,4)]);
            g.drawString("Hello world!", x, y);
        }
    }
}
```

getParameter  
accesses the values of  
the parameters

here, specify number  
of reps in Web page

uses getSize to  
get dimensions, pick  
random coords for text  
within the applet

```
<applet code="HelloWorld3.class" height=300 width=400>
  <param name="reps" value=200>
  You must use a Java-enabled browser to view this applet.
</applet>
```



# Applet Graphics

- in addition to displaying text
  - Dapat juga menggambar memperhitungkan Object Graphics

```
void drawLine(int x1, int y1, int x2, int y2)
```

```
void drawRect(int x, int y, int width, int height)
```

```
void fillRect(int x, int y, int width, int height)
```

```
void drawOval(int x, int y, int width, int height)
```

```
void fillOval(int x, int y, int width, int height)
```

- **EXAMPLE:** draw a red circle inscribed in a square, then draw random dots (dart pricks)
  - Dengan menghitung banyaknya jumlah titik di dalam vs. di luar lingkaran, dapat memperkirakan nilai dari  $\pi$

$$\pi = 4 * (\text{area of circle}/\text{area of square})$$



# Graphical Applet

```
public class Montel extends Applet
```

```
{  
    private static Random randy;  
    private int NUM_POINTS;  
    private int SIZE;
```

```
    private int RandomInRange(int low, int high) { CODE OMITTED }
```

```
    private double distance(int x1, int y1, int x2, int y2) { CODE OMITTED }
```

```
    public void init()
```

```
    {  
        randy = new Random();  
        NUM_POINTS = Integer.parseInt(getParameter("points"));  
        Dimension dim = getSize();  
        SIZE = Math.min(dim.width, dim.height);  
    }
```

```
    public void paint(Graphics g)
```

```
    {  
        g.setColor(Color.red);  
        g.fillOval(0, 0, SIZE, SIZE);  
        for (int i = 0; i < NUM_POINTS; i++) {  
            int x = RandomInRange(0, SIZE);  
            int y = RandomInRange(0, SIZE);  
            if (distance(x, y, SIZE/2, SIZE/2) < SIZE/2) {  
                g.setColor(Color.white);  
            }  
            else {  
                g.setColor(Color.black);  
            }  
            g.drawLine(x, y, x, y);  
        }  
    }
```

init method creates random number generator & gets parameters

paint method draws a circle and a bunch of random points

```
<applet code="Montel.class" height=300 width=300>  
    <param name="points" value=20000>  
    You must use a Java-enabled browser...  
</applet>
```



# Double Buffering

note: paint is called every time the page is brought to the front

- Pada versi saat ini dari Monte, hal ini berarti titik-titik baru digambar setiap saat setiap kali page tak dikenali dan kemudian membawanya dari belakang ke depan
  - *Membuang waktu menggambar ulang*
  - *Titik-titik berbeda setiap kali applet digambar*

the double buffering approach works by keeping an off-screen image

- **Pada init** method (yang dipanggil saat page di-load):
  - draw the figures on a separate, off-screen Graphics object*
- **Pada paint** method (yang dipanggil kapanpun page di tampilkan ke depan):
  - simply display the off-screen image on the screen*



```
public class Monte2 extends Applet
{
    private Image offScreenImage;
    private Graphics offScreenGraphics;
    ...
    public void init()
    {
        randy = new Random();
        NUM_POINTS = Integer.parseInt(getParameter("points"));
        Dimension dim = getSize();
        SIZE = Math.min(dim.width, dim.height);

        offScreenImage = createImage(SIZE, SIZE);
        offScreenGraphics = offScreenImage.getGraphics();

        offScreenGraphics.setColor(Color.red);
        offScreenGraphics.fillOval(0, 0, SIZE, SIZE);
        for (int i = 0; i < NUM_POINTS; i++) {
            int x = RandomInRange(0, SIZE);
            int y = RandomInRange(0, SIZE);
            if (distance(x, y, SIZE/2, SIZE/2) < SIZE/2) {
                offScreenGraphics.setColor(Color.white);
            }
            else {
                offScreenGraphics.setColor(Color.black);
            }
            offScreenGraphics.drawLine(x, y, x, y);
        }
    }
    public void paint(Graphics g)
    {
        g.drawImage(offScreenImage, 0, 0, null);
    }
}
```

# Buffered Applet

init method is called when page is loaded

does drawing to a separate, off-screen Graphics object

paint is called after init and whenever the applet is revisited

*Note: don't see image in progress*

```
<applet code="Monte2.class" height=300 width=300>
  <param name="points" value=20000>
  You must use a Java-enabled browser...
</applet>
```



```
public class Monte3 extends Applet
```

```
{  
  
public void init() {  
    randy = new Random();  
    NUM_POINTS = Integer.parseInt(getParameter("points"));  
    Dimension dim = getSize();  
    SIZE = Math.min(dim.width, dim.height);  
}  
  
public void paint(Graphics g) {  
    if (offScreenImage == null) {  
        offScreenImage = createImage(SIZE, SIZE);  
        offScreenGraphics = offScreenImage.getGraphics();  
  
        offScreenGraphics.setColor(Color.red);  
        g.setColor(Color.red);  
        offScreenGraphics.fillOval(0, 0, SIZE, SIZE);  
        g.fillOval(0, 0, SIZE, SIZE);  
        for (int i = 0; i < NUM_POINTS; i++) {  
            int x = randomInRange(0, SIZE);  
            int y = randomInRange(0, SIZE);  
            if (distance(x, y, SIZE/2, SIZE/2) < SIZE/2) {  
                offScreenGraphics.setColor(Color.white);  
                g.setColor(Color.white);  
            }  
            else {  
                offScreenGraphics.setColor(Color.black);  
                g.setColor(Color.black);  
            }  
            offScreenGraphics.drawLine(x, y, x, y);  
            g.drawLine(x, y, x, y);  
        }  
    }  
    else {  
        g.drawImage(offScreenImage, 0, 0, null);  
    }  
}  
}
```

# Better buffering

if want to see image as it is drawn, must be done in paint

when first loaded, have paint draw on the graphics screen and also to an off-screen buffer

on subsequent repaints, simply redraw the contents of the off-screen buffer

```
<applet code="Monte3.class" height=300 width=300>  
    <param name="points" value=20000>  
</applet>
```



# GUI Elements pada Applets

- Java mempunyai extensive library yang mendukung GUIs (Graphical User Interfaces)
  - Mempunyai element yang sesuai dengan HTML buttons, text boxes, text areas, ...
- Setiap element harus dibuat dan secara eksplisit ditambahkan ke applet

```
nameLabel = new Label("User's name");  
add(nameLabel);
```

```
nameField = new TextField(20);  
nameField.setValue("Dave");  
add(nameField);
```

- Java menyediakan beberapa class untuk mengontrol layout
  - `FlowLayout` adalah default
  - `BorderLayout` membolehkan penempatan element di sekitar borders applet
  - Sebuah `Panel` dapat berisi banyak element



# Text Boxes

```
public class Monte4 extends Applet
{
    private Label insideLabel;
    private TextField insideField;
    private Label outsideLabel;
    private TextField outsideField;

    public void init()
    {
        randy = new Random();
        NUM_POINTS =
        Integer.parseInt(getParameter("points"));
        Dimension dim = getSize();
        SIZE = Math.min(dim.width, dim.height);

        setLayout(new BorderLayout());

        Panel p = new Panel();
        insideLabel = new Label("Inside:");
        p.add(insideLabel);
        insideField = new TextField(5);
        p.add(insideField);
        outsideLabel = new Label("Outside:");
        p.add(outsideLabel);
        outsideField = new TextField(5);
        p.add(outsideField);

        add(p, BorderLayout.SOUTH);
    }
}
```

```
public void paint(Graphics g)
{
    . . .
    insideField.setText("0");
    outsideField.setText("0");
    . . .
    if (distance(x, y, SIZE/2, SIZE/2) <
        SIZE/2) {
        g.setColor(Color.white);
        int value =
        Integer.parseInt(insideField.getText())+1
        ;
        insideField.setText(""+value);
    }
    else {
        g.setColor(Color.black);
        int value =
        Integer.parseInt(outsideField.getText())+
        1;
        outsideField.setText(""+value);
    }
    . . .
}
}
```

[view page](#)

Pemrograman Web/TJ/AK045216/2 sks

```
<applet code="Monte4.class" height=335 width=300>
  <param name="points" value=20000>
</applet>
```



# Event Handling

in order to handle events (e.g., text changes, button clicks), can use the *event delegation model*

- Harus menentukan bahwa class meng-implementasikan the `ActionListener` interface

```
public class Monte5 extends Applet implements ActionListener
```

- Setiap source dari events harus didaftarkan dalam applet

```
dotButton = new Button("Click to generate dots");  
dotButton.addActionListener();
```

- Harus mempunyai sebuah `actionPerformed` method untuk menangani events

```
public void actionPerformed(ActionEvent e)  
{  
    if (e.getSource() == dotButton) {  
        drawDots();  
    }  
}
```



# ActionListener

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.util.Random;

public class Monte5 extends Applet
    implements ActionListener
{
    . . .
    private Button dotButton;

    public void init()
    {
        randy = new Random();
        NUM_POINTS =
            Integer.parseInt(getParameter("points"));
        Dimension dim = getSize();
        SIZE = dim.width;

        setLayout(new BorderLayout());
        dotButton =
            new Button("Click to generate dots");
        dotButton.addActionListener(this);
        add(dotButton, BorderLayout.SOUTH);

        drawCircle();
    }
}
```

```
public void drawCircle()
{
    CODE FOR DRAWING CIRCLE
}

public void drawDots()
{
    drawCircle();

    Graphics g = getGraphics();
    for (int i = 0; i < NUM_POINTS; i++) {
        CODE FOR DRAWING DOTS
    }
}

public void paint(Graphics g)
{
    g.drawImage(offScreenImage, 0, 0, null);
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == dotButton) {
        drawDots();
    }
}
}
```

[view page](#)

Pemrograman Web/TJ/AK045216/2 sks

```
<applet code="Monte5.class" height=325 width=300>
  <param name="points" value=20000>
</applet>
```