Oppgavesettet består av 3 oppgaver på 5 sider

Westerdals Oslo ACT

PG2100 – Programmering 2

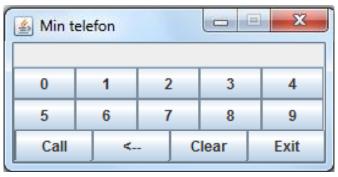
Tillatte hjelpemidler: ingen — Dato: 4.8.15
Vedlegg som kan være aktuelle: 12 (side 6 – 17) — Tid: 180 minutter

I alle oppgavene teller hvert delspørsmål likt dersom ikke annet er oppgitt.

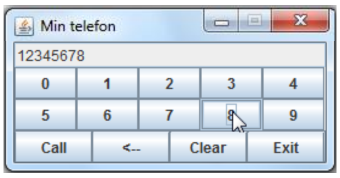
NB! Hvis du synes noe er uklart eller at opplysninger mangler, må du gjøre egne antagelser/forutsetninger, og løse oppgaven ut fra disse.

Oppgave 1 (40 %)

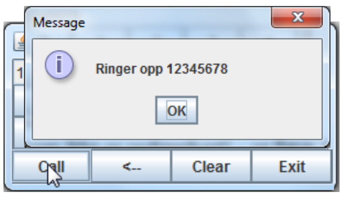
Figur 1 under viser brukergrensesnittet som en applikasjon setter opp når den starter.



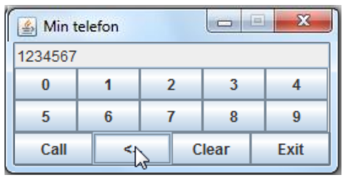
1



2



3



4

Applikasjonen simulerer tastaturet på en primitiv mobiltelefon.

Når brukeren klikker en av tallknappene, vises dette i tekstfeltet øverst (som ikke er redigerbart) – se figur 2.

Hvis bruker klikker **Call**, vises en melding om oppringing – se figur 3.

Hvis bruker klikker **<--**, viskes ett (og ett) tegn bort – se figur 4.

Hvis bruker klikker **Clear** viskes alle tegn bort, og brukergrensesnittet blir som ved oppstart (figur 1).

Hvis bruker klikker **Exit**, avsluttes applikasjonen.

Hvis **<--**, **Clear** eller **Call** klikkes når tekstfeltet er tomt, skal det ikke skje noe.

Vedlegg 12 viser delvis kode for denne applikasjonen.

Du skal skrive det som mangler på stedene markert som nummererte kommentarer.

Oppgave 2 (35 %)

a) (10 %) 1) Hva er riktig syntaks for å kalle på en superklasse-konstruktør fra en subklasse-konstruktør?

A. `super` etterfulgt av punktum (.)
B. `super` etterfulgt av et sett parenteser med argumenter til superklasse-konstruktøren
C. `super` etterfulgt av punktum og navnet på superklasse-konstruktøren
D. Ikke noe av det over

Velg riktig alternativ.

2) Hvilken Java-klasse arver *ikke* fra noen superklasse?

A. Integer
B. Object
C. String
D. Class

Velg riktig alternativ.

3) Hva skal vanligvis deklareres `private` i en klasse?

A. metoder
B. konstruktører
C. attributter
D. alle de over

Velg riktig alternativ.

4) Hva er *ikke* en superklasse/subklasse relasjon?

A. Avis/Aftenposten
B. Høgskole/Westerdals Oslo ACT
C. Seilbåt/Slepebåt
D. Land/Norge

Velg riktig alternativ.

5) Hvilken av følgende påstander er sann?

A. En klasse kan ikke ha flere konstruktører
B. Konstruktører har `void` som returtype
C. Konstruktører kan deklareres både med og uten parametere
D. En konstruktør kan ha et fritt valgt navn

Velg riktig alternativ.

(Oppgaven fortsetter på neste side)

b) (10 %) Studer følgende klasser (de ligger i samme fil):

```
 1  package oppgave2;
 2
 3  public class Oppgave2b {
 4     public static void main(String [] args) {
 5       ClassB b1 = new ClassB();
 6       System.out.println(b1.getIntA() + " " + b1.getIntB());
 7       ClassA a1 = new ClassA();
 8       ClassA a2 = b1;
 9       ClassB b2 = a1;
10     }
11  }
12
13  class ClassA {
14    private int intA = 0;
15
16    public ClassA() {
17      intA = 7;
18    }
19
20    public int getIntA () {
21      return intA;
22    }
23  }
24
25  class ClassB extends ClassA {
26    private int intB = -1;
27
28    public ClassB() {
29      intB = 8;
30    }
31
32    public int getIntB () {
33      return intB;
34    }
35  }
```

1) Hvilken linje vil kompilatoren gi feilmelding om? Hva skyldes feilen?

Anta at du kommenterer vekk linjen som kompilatoren melder feil om.

2) Hva blir output når programmet blir utført?

(Oppgaven fortsetter på neste side)

c) (15 %) En `static` metode `update` har følgende parametere:

en `ArrayList` av `BankAccount`
en `BankAccount`

Klassen `BankAccount` (bankkonto) er gitt i vedlegg 11.

Metoden sjekker om listen inneholder en konto med samme navn som den bestemte bankkontoen (den andre parameteren).
Hvis listen inneholder en slik konto, ***erstattes*** den med den bestemte kontoen, ellers ***legges*** denne kontoen inn i listen.

Metoden returnerer ikke noe.

Eksempler:
Anta at listen `bankListe` i utgangspunktet inneholder tre kontoer med følgende innhold:

```
AKonto, 100.0
BKonto, 200.0
CKonto, 300.0
```

Så gjøres følgende kall:

```
update(bankListe, konto);
```

der `konto` inneholder følgende data:

```
Bkonto, 2000.0
```

Innholdet i listen etterpå skal da være:

```
AKonto, 100.0
BKonto, 2000.0
CKonto, 300.0
```

Så kalles metoden `update` med *denne* listen og følgende bankkonto som argumenter:

```
Dkonto, 5000.0
```

Innholdet i listen etterpå skal da være:

```
AKonto, 100.0
BKonto, 2000.0
CKonto, 300.0
Dkonto, 5000.0
```

Skriv metoden `update`.

Oppgave 3 (25 %)

Følgende klasser er deklarert i samme fil:

```
public class Oppgave3 {
 public static void main(String [] args) {
  Lamp[] elements =
   {
    new Book(),
    new Pen(),
    new Lamp(),
    new Sock()
   };

  for (int i = 0; i < elements.length; i++) {
   System.out.println(elements[i]);
   elements[i].method1();
   elements[i].method2();
   System.out.println();
  }
 }
```

```
class Lamp {
 public void method1() {
  System.out.println("lamp 1");
 }

 public void method2() {
  System.out.println("lamp 2");
 }

 public String toString() {
  return "lamp";
 }
}

class Sock extends Lamp {
 public void method1() {
  System.out.println("sock 1");
 }

 public String toString() {
  return "sock";
 }
}

class Book extends Sock {
 public void method2() {
  System.out.println("book 2");
 }
}

class Pen extends Sock {
 public void method1() {
  System.out.println("pen 1");
```

Vis hvordan output blir når programmet `Oppgave3` blir kjørt.

--- Slutt på oppgavesettet ---

Vedlegg

Vedlegg 1 Class ArrayList (utdrag)

Method Summary	
`boolean`	`add(E o)` Appends the specified element to the end of this list.
`void`	`add(int index, E element)` Inserts the specified element at the specified position in this list.
`boolean`	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
`boolean`	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified Collection into this list, starting at the specified position.
`void`	`clear()` Removes all of the elements from this list.
`Object`	`clone()` Returns a shallow copy of this `ArrayList` instance.
`boolean`	`contains(Object elem)` Returns `true` if this list contains the specified element.
`E`	`get(int index)` Returns the element at the specified position in this list.
`boolean`	`isEmpty()` Tests if this list has no elements.
`int`	`lastIndexOf(Object elem)` Returns the index of the last occurrence of the specified object in this list.
`E`	`remove(int index)` Removes the element at the specified position in this list.
`boolean`	`remove(Object o)` Removes a single instance of the specified element from this list, if it is present (optional operation).
`E`	`set(int index, E element)` Replaces the element at the specified position in this list with the specified element.
`int`	`size()` Returns the number of elements in this list.
`void`	`trimToSize()` Trims the capacity of this `ArrayList` instance to be the list's current size.

Vedlegg 2 Class String

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared.

Constructor Summary
`String``()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String``(byte[] bytes)` Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.
`String``(String original)` Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

Method Summary	
`char`	**`charAt`**`(int index)` Returns the `char` value at the specified index.
`int`	**`codePointAt`**`(int index)` Returns the character (Unicode code point) at the specified index.
`int`	**`codePointBefore`**`(int index)` Returns the character (Unicode code point) before the specified index.
`int`	**`codePointCount`**`(int beginIndex, int endIndex)` Returns the number of Unicode code points in the specified text range of this `String`.
`int`	**`compareTo`**`(String anotherString)` Compares two strings lexicographically.
`int`	**`compareToIgnoreCase`**`(String str)` Compares two strings lexicographically, ignoring case differences.
`String`	**`concat`**`(String str)` Concatenates the specified string to the end of this string.
`boolean`	**`contains`**`(CharSequence s)` Returns true if and only if this string contains the specified sequence of char values.
`boolean`	**`contentEquals`**`(CharSequence cs)` Returns `true` if and only if this `String` represents the same sequence of char values as the specified sequence.
`boolean`	**`contentEquals`**`(StringBuffer sb)` Returns `true` if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`.
`static String`	**`copyValueOf`**`(char[] data)` Returns a String that represents the character sequence in the array specified.
`static String`	**`copyValueOf`**`(char[] data, int offset, int count)` Returns a String that represents the character sequence in the array specified.
`boolean`	**`endsWith`**`(String suffix)` Tests if this string ends with the specified suffix.
`boolean`	**`equals`**`(Object anObject)` Compares this string to the specified object.
`boolean`	**`equalsIgnoreCase`**`(String anotherString)` Compares this `String` to another `String`, ignoring case considerations.
`static String`	**`format`**`(Locale l, String format, Object... args)` Returns a formatted string using the specified locale, format string, and arguments.
`static String`	**`format`**`(String format, Object... args)` Returns a formatted string using the specified format string and arguments.
`byte[]`	**`getBytes`**`()` Encodes this `String` into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
`void`	**`getBytes`**`(int srcBegin, int srcEnd, byte[] dst, int dstBegin)` **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the* `getBytes()` *method, which uses the platform's default charset.*
`byte[]`	**`getBytes`**`(String charsetName)` Encodes this `String` into a sequence of bytes using the named charset, storing the result into a new byte array.
`void`	**`getChars`**`(int srcBegin, int srcEnd, char[] dst, int dstBegin)` Copies characters from this string into the destination character array.
`int`	**`hashCode`**`()` Returns a hash code for this string.

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the rightmost occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex) Tells whether or not this string matches the given regular expression.
int	**offsetByCodePoints**(int index, int codePointOffset) Returns the index within this `String` that is offset from the given `index` by `codePointOffset` code points.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
String	**replace**(CharSequence target, CharSequence replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
String	**replaceAll**(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String	**replaceFirst**(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[]	**split**(String regex) Splits this string around matches of the given regular expression.
String[]	**split**(String regex, int limit) Splits this string around matches of the given regular expression.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if this string starts with the specified prefix beginning a specified index.
CharSequence	**subSequence**(int beginIndex, int endIndex) Returns a new character sequence that is a subsequence of this sequence.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this `String` to lower case using the rules of the default locale.
String	**toLowerCase**(Locale locale) Converts all of the characters in this `String` to lower case using the rules of the given `Locale`.
String	**toString**() This object (which is already a string!) is itself returned.

String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.
String	**toUpperCase**(Locale locale) Converts all of the characters in this String to upper case using the rules of the given Locale.
String	**trim**() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static String	**valueOf**(char[] data) Returns the string representation of the char array argument.
static String	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	**valueOf**(double d) Returns the string representation of the double argument.
static String	**valueOf**(float f) Returns the string representation of the float argument.
static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.
static String	**valueOf**(Object obj) Returns the string representation of the Object argument.

Vedlegg 3 Class JFrame

The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. This is different from the AWT Frame case. As a conveniance add and its variants, remove and setLayout have been overridden to forward to the contentPane as necessary. This means you can write:

```
        frame.add(child);
```

And the child will be added to the contentPane. The content pane will always be non-null. Attempting to set it to null will cause the JFrame to throw an exception. The default content pane will have a BorderLayout manager set on it. Refer to RootPaneContainer for details on adding, removing and setting the LayoutManager of a JFrame.

Unlike a Frame, a JFrame has some notion of how to respond when the user attempts to close the window. The default behavior is to simply hide the JFrame when the user closes the window. To change the default behavior, you invoke the method setDefaultCloseOperation(int). To make the JFrame behave the same as a Frame instance, use setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE).

Field Summary	
static int	**EXIT_ON_CLOSE** The exit application default window close operation.
Fields inherited from class java.awt.Component	
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT	
Fields inherited from interface javax.swing.WindowConstants	
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE	

Constructor Summary
JFrame() Constructs a new frame that is initially invisible.
JFrame(String title) Creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc) Creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Method Summary	
Container	**getContentPane**() Returns the contentPane object for this frame.
protected void	**processWindowEvent**(WindowEvent e)

	Processes window events occurring on this component.
void	**remove**(Component comp) Removes the specified component from the container.
void	**setContentPane**(Container contentPane) Sets the `contentPane` property.
void	**setDefaultCloseOperation**(int operation) Sets the operation that will happen by default when the user initiates a "close" on this frame.
void	**setGlassPane**(Component glassPane) Sets the `glassPane` property.
void	**setIconImage**(Image image) Sets the image to be displayed in the minimized icon for this frame.
void	**setJMenuBar**(JMenuBar menubar) Sets the menubar for this frame.
void	**setLayeredPane**(JLayeredPane layeredPane) Sets the `layeredPane` property.
void	**setLayout**(LayoutManager manager) Sets the `LayoutManager`.
protected void	**setRootPane**(JRootPane root) Sets the `rootPane` property.
protected void	**setRootPaneCheckingEnabled**(boolean enabled) Sets whether calls to `add` and `setLayout` are forwarded to the `contentPane`.
void	**update**(Graphics g) Just calls `paint(g)`.

Vedlegg 4 Class JPanel

`JPanel` is a generic lightweight container. For examples and task-oriented documentation for JPanel, see How to Use Panels, a section in *The Java Tutorial*.
Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans^{TM} has been added to the `java.beans` package. Please see `XMLEncoder`.

Fields inherited from class javax.swing.JComponent
accessibleContext, listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Constructor Summary
JPanel() Creates a new `JPanel` with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered) Creates a new `JPanel` with `FlowLayout` and the specified buffering strategy.
JPanel(LayoutManager layout) Create a new buffered JPanel with the specified layout manager
JPanel(LayoutManager layout, boolean isDoubleBuffered) Creates a new JPanel with the specified layout manager and buffering strategy.

Method Summary	
AccessibleContext	**getAccessibleContext**() Gets the AccessibleContext associated with this JPanel.
PanelUI	**getUI**() Returns the look and feel (L&F) object that renders this component.
String	**getUIClassID**() Returns a string that specifies the name of the L&F class that renders this component.
protected String	**paramString**() Returns a string representation of this JPanel.
void	**setUI**(PanelUI ui) Sets the look and feel (L&F) object that renders this component.

void	**updateUI**() Resets the UI property with a value from the current look and feel.

Vedlegg 5 Interface ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.

Method Summary	
void	**actionPerformed**(ActionEvent e) Invoked when an action occurs.

Vedlegg 6 Class ActionEvent

A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a `Button`) when the component-specific action occurs (such as being pressed). The event is passed to every every `ActionListener` object that registered to receive such events using the component's `addActionListener` method.
Note: To invoke an `ActionEvent` on a `Button` using the keyboard, use the Space bar.
The object that implements the `ActionListener` interface gets this `ActionEvent` when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

Field Summary	
static int	**ACTION_FIRST** The first number in the range of ids used for action events.
static int	**ACTION_LAST** The last number in the range of ids used for action events.
static int	**ACTION_PERFORMED** This event id indicates that a meaningful action occured.
static int	**ALT_MASK** The alt modifier.
static int	**CTRL_MASK** The control modifier.
static int	**META_MASK** The meta modifier.
static int	**SHIFT_MASK** The shift modifier.

Constructor Summary
ActionEvent(Object source, int id, String command) Constructs an `ActionEvent` object.
ActionEvent(Object source, int id, String command, int modifiers) Constructs an `ActionEvent` object with modifier keys.
ActionEvent(Object source, int id, String command, long when, int modifiers) Constructs an `ActionEvent` object with the specified modifier keys and timestamp.

Method Summary	
String	**getActionCommand**() Returns the command string associated with this action.
int	**getModifiers**() Returns the modifier keys held down during this action event.
long	**getWhen**() Returns the timestamp of when this event occurred.
String	**paramString**() Returns a parameter string identifying this action event.

Vedlegg 7 Class JButton

An implementation of a "push" button

Constructor Summary
JButton()   Creates a button with no set text or icon.
JButton(Action a)   Creates a button where properties are taken from the `Action` supplied.
JButton(Icon icon)   Creates a button with an icon.
JButton(String text)   Creates a button with text.
JButton(String text, Icon icon)   Creates a button with initial text and an icon.

Method Summary	
AccessibleContext	**getAccessibleContext**() Gets the `AccessibleContext` associated with this `JButton`.
String	**getUIClassID**() Returns a string that specifies the name of the L&F class that renders this component.
boolean	**isDefaultButton**() Gets the value of the `defaultButton` property, which if `true` means that this button is the current default button for its `JRootPane`.
boolean	**isDefaultCapable**() Gets the value of the `defaultCapable` property.
protected String	**paramString**() Returns a string representation of this `JButton`.
void	**removeNotify**() Overrides `JComponent.removeNotify` to check if this button is currently set as the default button on the `RootPane`, and if so, sets the `RootPane`'s default button to `null` to ensure the `RootPane` doesn't hold onto an invalid button reference.
void	**setDefaultCapable**(boolean defaultCapable) Sets the `defaultCapable` property, which determines whether this button can be made the default button for its root pane.
void	**updateUI**() Resets the UI property to a value from the current look and feel.

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, fireVetoableChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentGraphics, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintChildren, paintComponent, paintImmediately, paintImmediately, print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent, processKeyBinding, processKeyEvent, processMouseEvent, processMouseMotionEvent, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setUI, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Vedlegg 8 Class Color

The `Color` class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a `ColorSpace`. Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor. The alpha value defines the transparency of a color and can be represented by a float value in the range 0.0 - 1.0 or 0 - 255. An alpha value of 1.0 or 255 means that the color is completely opaque and an alpha value of 0 or 0.0 means that the color is completely transparent. When constructing a `Color` with an explicit alpha or getting the color/alpha components of a `Color`, the color components are

Field Summary	
static Color	**black** The color black.
static Color	**BLACK** The color black.
static Color	**blue** The color blue.
static Color	**BLUE** The color blue.
static Color	**cyan** The color cyan.
static Color	**CYAN** The color cyan.
static Color	**DARK_GRAY** The color dark gray.
static Color	**darkGray** The color dark gray.
static Color	**gray** The color gray.
static Color	**GRAY** The color gray.
static Color	**green** The color green.
static Color	**GREEN** The color green.
static Color	**LIGHT_GRAY** The color light gray.
static Color	**lightGray** The color light gray.
static Color	**magenta** The color magenta.
static Color	**MAGENTA** The color magenta.
static Color	**orange** The color orange.
static Color	**ORANGE** The color orange.
static Color	**pink** The color pink.
static Color	**PINK** The color pink.
static Color	**red** The color red.
static Color	**RED** The color red.
static Color	**white** The color white.
static Color	**WHITE** The color white.
static Color	**yellow** The color yellow.
static Color	**YELLOW** The color yellow.

Vedlegg 9 Class BorderLayout

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: `NORTH`, `SOUTH`, `EAST`, `WEST`, and `CENTER`. When adding a component to a container with a border layout, use one of these five constants.

The components are laid out according to their preferred sizes and the constraints of the container's size. The `NORTH` and `SOUTH` components may be stretched horizontally; the `EAST` and `WEST` components may be stretched vertically; the `CENTER` component may stretch both horizontally and vertically to fill any space left over.

Field Summary	
static String	**AFTER_LAST_LINE** Synonym for PAGE_END.
static String	**AFTER_LINE_ENDS** Synonym for LINE_END.
static String	**BEFORE_FIRST_LINE** Synonym for PAGE_START.
static String	**BEFORE_LINE_BEGINS** Synonym for LINE_START.
static String	**CENTER** The center layout constraint (middle of container).
static String	**EAST** The east layout constraint (right side of container).
static String	**LINE_END** The component goes at the end of the line direction for the layout.
static String	**LINE_START** The component goes at the beginning of the line direction for the layout.
static String	**NORTH** The north layout constraint (top of container).
static String	**PAGE_END** The component comes after the last line of the layout's content.
static String	**PAGE_START** The component comes before the first line of the layout's content.
static String	**SOUTH** The south layout constraint (bottom of container).
static String	**WEST** The west layout constraint (left side of container).

Constructor Summary
BorderLayout() Constructs a new border layout with no gaps between components.
BorderLayout(int hgap, int vgap) Constructs a border layout with the specified gaps between components.

Vedlegg 10 Class GridLayout

The `GridLayout` class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

When both the number of rows and the number of columns have been set to non-zero values, either by a constructor or by the `setRows` and `setColumns` methods, the number of columns specified is ignored. Instead, the number of columns is determined from the specified number of rows and the total number of components in the layout. So, for example, if three rows and two columns have been specified and nine components are added to the layout, they will be displayed as three rows of three columns. Specifying the

Constructor Summary
GridLayout() Creates a grid layout with a default of one column per component, in a single row.
GridLayout(int rows, int cols) Creates a grid layout with the specified number of rows and columns.
GridLayout(int rows, int cols, int hgap, int vgap) Creates a grid layout with the specified number of rows and columns.

Method Summary	
void	**addLayoutComponent**(String name, Component comp) Adds the specified component with the specified name to the layout.

int	getColumns() Gets the number of columns in this layout.
int	getHgap() Gets the horizontal gap between components.
int	getRows() Gets the number of rows in this layout.
int	getVgap() Gets the vertical gap between components.
void	layoutContainer(Container parent) Lays out the specified container using this layout.
Dimension	minimumLayoutSize(Container parent) Determines the minimum size of the container argument using this grid layout.
Dimension	preferredLayoutSize(Container parent) Determines the preferred size of the container argument using this grid layout.
void	removeLayoutComponent(Component comp) Removes the specified component from the layout.
void	setColumns(int cols) Sets the number of columns in this layout to the specified value.
void	setHgap(int hgap) Sets the horizontal gap between components to the specified value.
void	setRows(int rows) Sets the number of rows in this layout to the specified value.
void	setVgap(int vgap) Sets the vertical gap between components to the specified value.
String	toString() Returns the string representation of this grid layout's values.

Vedlegg 11 Class BankAccount

```
public class BankAccount {
   private String name;
   private double balance;

   public BankAccount() {
   }

   public BankAccount(String name, double balance) {
      this.name = name;
      this.balance = balance;
   }

   public String getName() {
      return name;
   }

   public void deposit(double amount) { //sette inn penger på kontoen
      balance += amount;
   }

   public void withdraw(double amount) {  //ta ut penger fra kontoen
      if (balance >= amount) {
         balance = balance - amount;
      }
   }

   public String toString() {
      return name + ", " + balance;
   }
```

(fortsetter på neste side)

```java
   public boolean equals(Object other) {
      if (!(other instanceof BankAccount)) return false;
      if (other == this) return true;
      BankAccount b = (BankAccount) other;
      return this.name.equals(b.name);
   }
  }
```

Vedlegg 12 Class Phone

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

public class Phone extends JFrame implements ActionListener {
   private JButton[] talltaster;    // knapper for tallene
   private JButton[] valgtaster;    // knapper for valgene
   private JTextField display;      // tekstfelt for nummervisning
   private JPanel pnlTall;          // panel for tallene
   private JPanel pnlValg;          // panel for valgene

   public Phone() {
      setTitle("Min telefon");

      // 1 – panel og array for talltaster opprettes her


      pnlValg = new JPanel(new GridLayout(1, 4));
      valgtaster = new JButton[4];
      valgtaster[0] = new JButton("Call");
      valgtaster[1] = new JButton("<--");
      valgtaster[2] = new JButton("Clear");
      valgtaster[3] = new JButton("Exit");
      for (int i = 0; i < 4; i++) {
         pnlValg.add(valgtaster[i]);
         valgtaster[i].addActionListener(this);
      }

      // 2 – plassering/oppretting av komponenter og standard oppsett av vindu gjøres her

   }

   public void actionPerformed(ActionEvent e) {

   // 3 – handlinger for klikk på de ulike knappene gjøres her

   }

   public static void main(String arg[]) {
      new Phone();
   }
}
```