Application Development Module 2

Swing Layout Managers

A Java AWT object that automatically defines the position of widgets within a containere for you. Most Layouts will override any position you set, deffering to their own calculated position instead.

FlowLayout

The default layout a JPanel has.

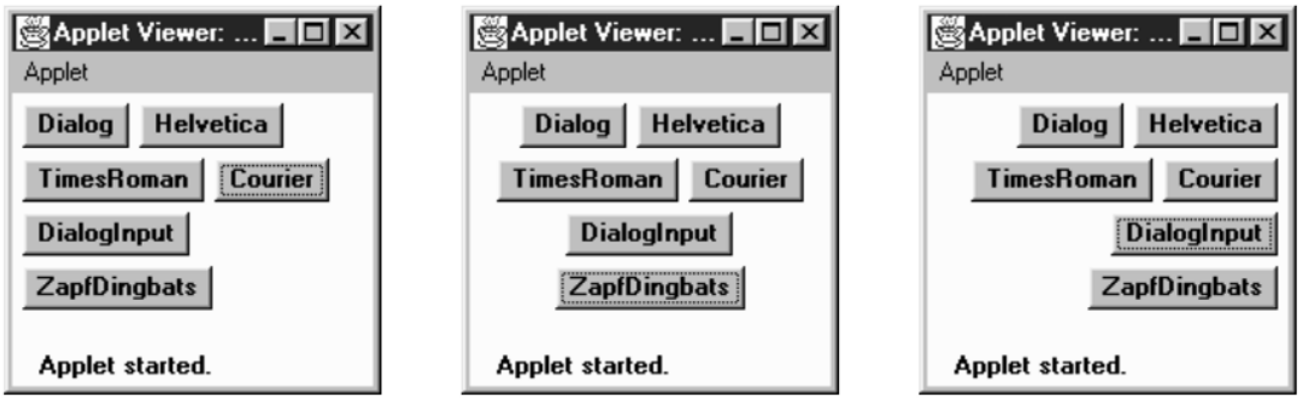
This layout arranges components in order, from left to right. If the container cannot fit the component, it will be added in the next row at the bottom.

The layout automatically repositions the components when the container resizes.

Components can have a preferred size, taking available space.

If there are too many componenets and too little space, the components can be clipped and you wont see all of it.

A Flowlayout can prefer to align things, simply pass this in the constructor.



By default, the horizontal and vertical gap is **5 pixels**, unless specified also in the constructor.

Constructor summary

```
FlowLayout(int alignment, int hgap, int vgap)
```

BorderLayout

The default for a top-level JWindow or JFrame.

> A JFrame or JWindow is a container that provides a window interface to interact with. JPanels on the other hand are merely logical containers.

The BorderLayout has 5 regions, North, South, West, East and Center.

Components can be added to any of those regions, unused space from other regions can be given up for another region.

Components inside tend to fill all the space of the region.

You can add multiple components to a single region, however only the latest one will be actually shown.

Using `add()` requires a constraint for the region defined in the BorderLayout Class.

```
myBorderLayout.add(component, BorderLayout.SOUTH);
```

Constructor summary

```
BorderLayout(int hgap, int vgap)
```

GridLayout

A layout that puts objects into rows and columns, each cell having equal size.

Using `add()` will add the component from left to right, and top to bottom cells. There is no feature to put in a specific cell.

You can specify the grid dimension in the constructor.

Setting 0 for a row or column will allow the layout to grow in that direction with no bound. However Specifying 0 for both rows and columns will throw a `IllegalArgumentException`.

Constructor summary

```
GridLayout(int rows, int cols, int hgap, int vgap)
```

GridBagLayout

Probably the most versatile layout you should be using.

Elements are arranged in a grid similar to the `GridLayout`. But elements can have different spans of cells.

Additionally unlike `BorderLayout` or `GridLayout`, the GridBagLayout sizes components to their preferred sizes rather than trying to fill up all space in the region.

The properties of the widget are defined with a `GridBagConstraints` object.

Constructor Summary

```
GridbagLayout()
```

Adjusting widget properties with gridbag constraints

This is probably one downside of the gridbag layout, its tedious.

First you need a grid bag constraints object.

Constructor Summary

```
GridBagConstraints(int gridx,
                           int gridy,
                           int gridwidth,
                           int gridheight,
                           double weightx,
                           double weighty,
                           int anchor,
                           int fill,
                           Insets insets,
                           int ipadx,
                           int ipady)
```

This constructor is tedious, you can also provide nothing and configure each parameter manually.

```
GridBagConstraints()
```

Parameters to configure

- gridx, The position on the grid horizontally.
- gridy, The position on the vertically.
- gridwidth, how many cells the component wil ltake horizontally.
- gridheight, how many cells the component will take vertically.
- weightx, how much extra space it can get from empty spaces horizontally.
- weighty, how much extra space it can get from empty spaces vertically.
- anchor, what side the component should prefer to stay at.
- fill, How a component should take up textra space in its cell, be default it is `GridBagConstraints.NONE`.
- insets, Specifies external padding.
- ipadx, Specifies internal padding horizontally.

- ipady, Specifies internal padding vertically.

Insets constructor Summary

```
Insets(int top, int left, int bottom, int right)
```

> Regarding weightx and weighty. A component with 0 weight will simply stay in its own cell and not take up any more grid boxes. A component by default has a weight of 1, it will take up any remaining free space from other cells.

CardLayout

A layout capable of switching components.

With CardLayout, there is no space between components because only one component is visible at a time. The gaps can be thought of as Insets instead.

When adding a Layout to the `CardLayout`, the `add()` functions needs another parameter. you need to provide the jpanel as usual, and you ned to provide a name in `String`.

```
cardLayout.add(component, name)
```

Common methods in CardLayout

- `next(Container parent)`, flip to the next card
- `previous(Container parent)`, flip to the previous card
- `first(Container parent)`, flip to the first card
- `last(Container parent)`, flip to the last added card
- `show(Container parent, String name)`, flip to a specific card using the name

Constructor Summary

```
CardLayout (int hgap, int vgap)
```

Principles of Designing UI

- Functional based form. The appearance of an object should reflect or allude to its purpose. (Eg: A disabled button should be grayed out and flat)
- Visibility. It should be easy for a user to find functions in a program, too many elements at once may make it challenging.
- Strong Mapping,

- Feeedback. The system should provide information about their action, for example buttons become dented when pressed, or make a sound when clicked.
- Affordance
- Consistency, elements should be designed with consistency to make it comfortable, and to help the user let inutition guide them.
- Forgiveness, software should include features to help user avoid or recover from mistakes. (Eg: A confirmation prompt, auto saving, etc.)
- Minimalism, a deliberate reduction in complexity for the user's sake

Dueling Principles

Some principles tend to clash, or it might not be possible to have all principles. A compromise must be made based on what the user needs.

Fitts' law

It states that the time to reach an element with the cursor depends on the distance and size of the element.

Larger buttons are much easier to click than smaller ones. And the position of the element relative to the cursor is important.

Java Persistence API

An open source API made by Oracle for accessing and managing data of the database as java objects.

It defines a object-relational mapping (ORM) a standard method for mapping java classes to their database counterpart.

Classes can be mapped to database objects through annotations. Frequentely this information is stored in the meta information file.

It is not strictly 1 to 1. A java entity can be mapped to a set of columns for more than one table.

Steps to register a java entity with the ORM

1. Import the persistence API

```
import javax.persistence.*;
```

2. Create an entity manager factory

```
EntityManagerFactory mangerFactory = new EntityManagerFactory("db_name");
```

3. Obtain the entity manager

```
EntityManager entityManager = managerFactory.createEntityManager();
```

4. Initialize the entity manager

```
entityManager.getTransaction().begin();
```

5. Provide the new data through your java entity

```
Student student1 = new Student();
student1.setId(1);
student1.setName("John");
```

6. Persist the object

```
entittyManager.persist(student1);
```

7. Close up the transaction and release the factory resources

```
entityManager.close();
managerFactory.close();
```