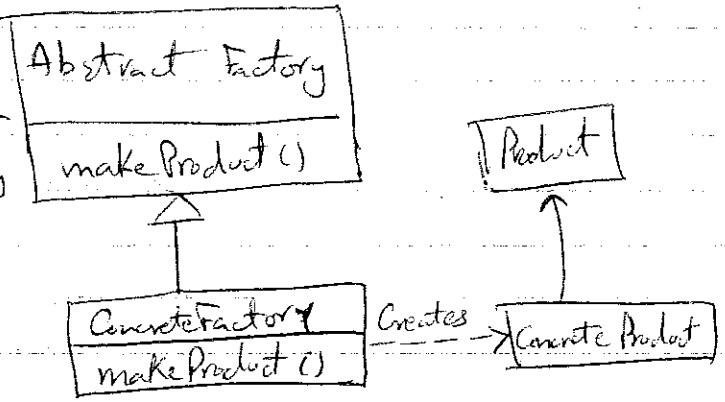
#1) Factory Pattern:

Category: Creational
Intent: To define an interface for creating objects but let subclasses decided which class to instantiate & how.

Applicability: Should be used when a system should be independent of how its Products are created.

Example ~~usage, design~~:
↳ used in sorting Program to create concrete sorting Algorithems.



Factory Method Pattern:

Category: Creational
intent: Define an interface for creating an object but defer instantiation to subclass.

Applicability:
- used when a class cannot anticipate the class of objects it must create
- when a class defers to its subclasses to specify the objects it creates.

ex) used in Scribble Pad Program

Abstract Factory.

Category: Creational
Intent: To Provide an interface for creating a family of Related or ~~dependent~~ dependent objects w/o specifying their concrete classes.

Applicability:
- when a system should be independent of how it's components are created.
- when a system should be configurable with one of Multiple interchangeable families of Products
- when a family of Related Products should not be mixed with similar Products from diff families.

Ex) in the maze game, defines the Responsibilities for creating the Parts & building the maze based using the Parts.

Also used for Theming

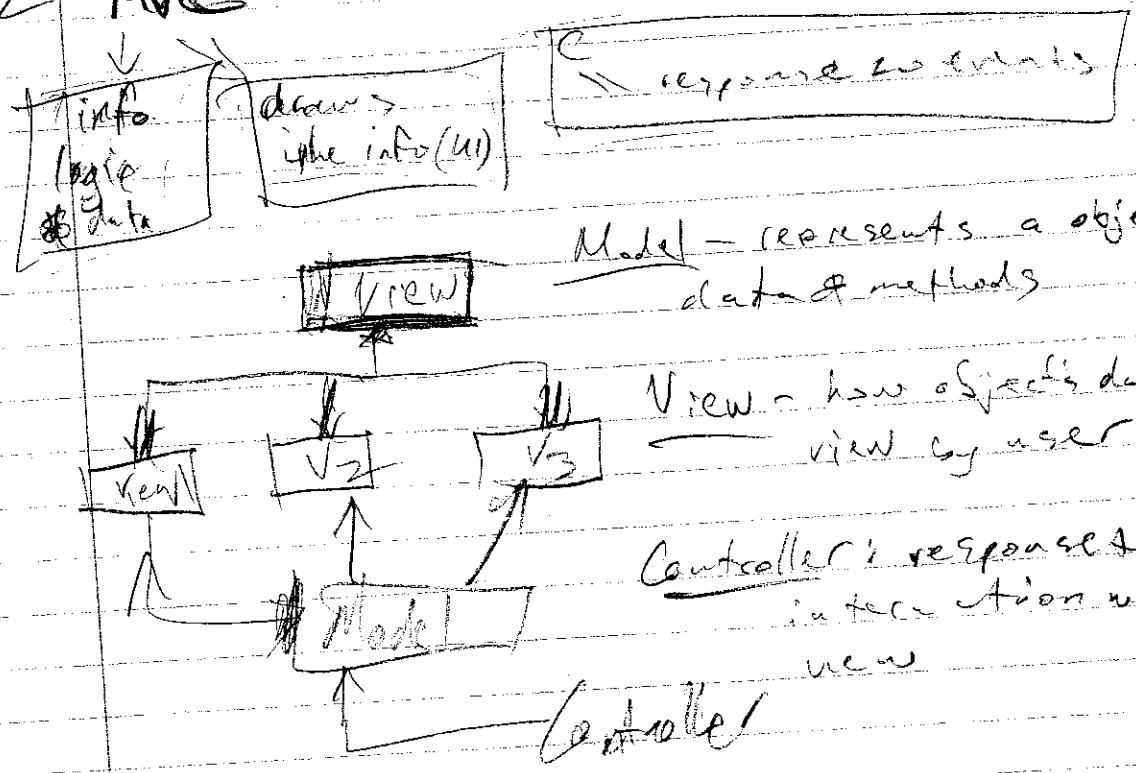
Lect 22

#9 Threads

Contentions: mechanism to run
when threads of higher
priority are always running
or same priority never yields

Deadlocks: threads blocking each other
and none get to run
b/c compete for multiple resources

#2 MVC Architect pattern



Model - represents a object's
data & methods

View - how objects data are
view by user

Controller: responses to user's
interaction w/ objects
view

3) Give a short code fragment that makes use of Java 1.5 enumerated types.

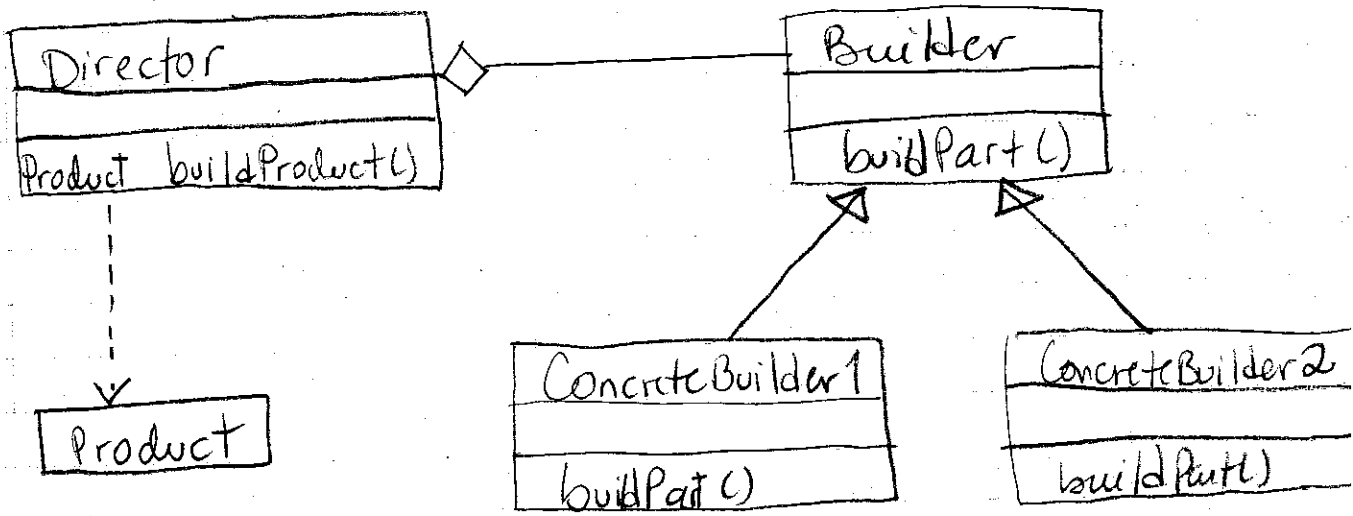
```
public enum Movement
{
   UP, DOWN, LEFT, RIGHT
}

//driver class can call one of its methods
Movement m = Movement.valueOf("UP");
```

#4) Designer pattern : Builder

Use this pattern in video games to avoid the duplication of similar code used in different levels.
This pattern is used to separate the construction of a complex object (the video game) from the implementation of its parts so that the same construction process can create complex objects from different implementations of parts.

The structure of the Builder pattern is shown in the following diagram:



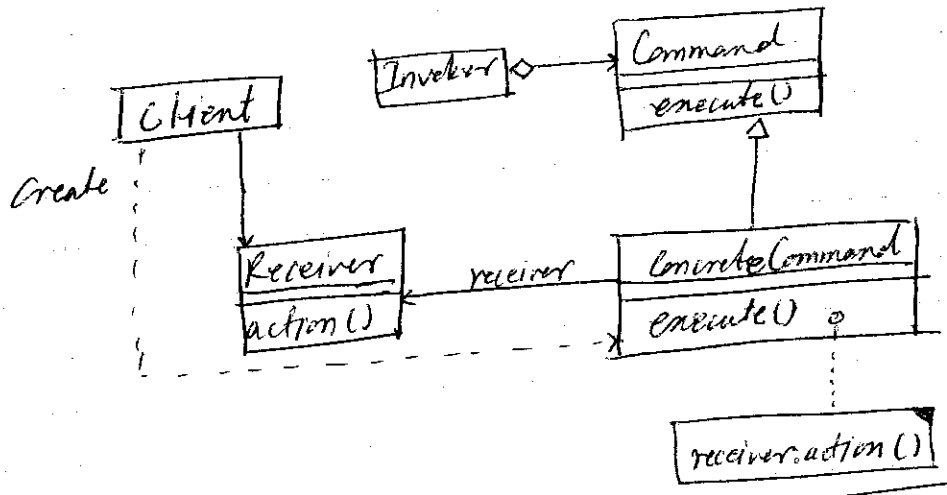
GYANENDRA RYAN LAMA

EXTRA POINT

Design Pattern Command

This pattern is very useful in ~~[illegible]~~ creating programs like word processor when an undo feature might be very popular.

This is a behavioral design pattern



For Example

Class maze.UndoableCommand

```
package maze;
public interface UndoableCommand extends Command
{
    public void undo();
}
```

Undoable moves in the maze game using the command pattern.

Class maze.Command

```
package maze;
public interface Command {
    public void execute();
}
```

Client creates the concrete command and binds the concrete command to their receivers.

6) How would you create a Thread that uses a Runnable, and then start that Thread?

page 553:

```
public class Counter2 implements Runnable
{
   ---

   public Counter2(---)
   {
      ---
   }

   public void run()
   {
      ---
   }

   public static void main(String[] args)
   {
      new Thread(new Counter2(0, 1, 33)).start();
      ---
   }
}
```

ANSWER

ANSWER

Number 7

```
public synchronized int get()
    {
    while (available == false)
        {
            try
            { //
                // wait for Producer to put value (Producer consumer)
                wait();
            }
            catch (InterruptedException e) { }
        }
        available = false;
        // notify Producer that value has been retrieved
        notifyAll();
        return contents;
    }
```

Until there get method loops until the producer has produced a new value.
get calls the wait method.

```
public synchronized void put(int value) {
    { while (available == true)
        { try
            { // wait for consumer to get value
              wait();
            }
            catch (InterruptedException e) {}
        }
        contents = value;
        available = true;
        // notify consumer that value has been set
        notifyAll();
    }
```

Bank ATM example:

```
public class BankATM{
    public static int accountAmt = 0;
    ...
    public static void deposit(int amount){
        if(amount < 0){
            Throw new Exception("Cannot deposit a negative amount.");
        }
        else{
            Synchronized(this){
                accountAmt += deposit;
            }
        }
    }
    ...
}
```

8.

- Synchronized keyword defines a function or block of code in which only one thread can enter at a time. This enforces that multiple threads cannot change the state of an object at the same time and therefore corrupt the object data.

- In our example, the step where we are adding the deposit to the account amount is synchronized so that two deposits to the account at the same time will not corrupt the accountAmount object.

#9

ⓐ Contention:

I) Occurs when a runnable thread never gets a chance to run.

II) Occurs when one or more threads of higher priority are always running and do not allow threads of lower priority to run.

III) Occurs when ● threads of the same priority are in the runnable state, one of them is running but never yields.

ⓑ Deadlock

I Two or more threads block each other and none proceed.

I Often because of competition for multiple shared resources that the threads require exclusive possession of simultaneously.

```
public class MultiEchoServer
{
        public static void main(String[] args)
        {
        try
        {
                ServerSocket s= new ServerSocket(8009);
                while(true)
                {
                        Socket incoming = s.accept();
                        new ClientHandler(incoming).start();
                        //ClientHandler extends Thread
                }
        }
        catch(Exception e){}
        }
}
```