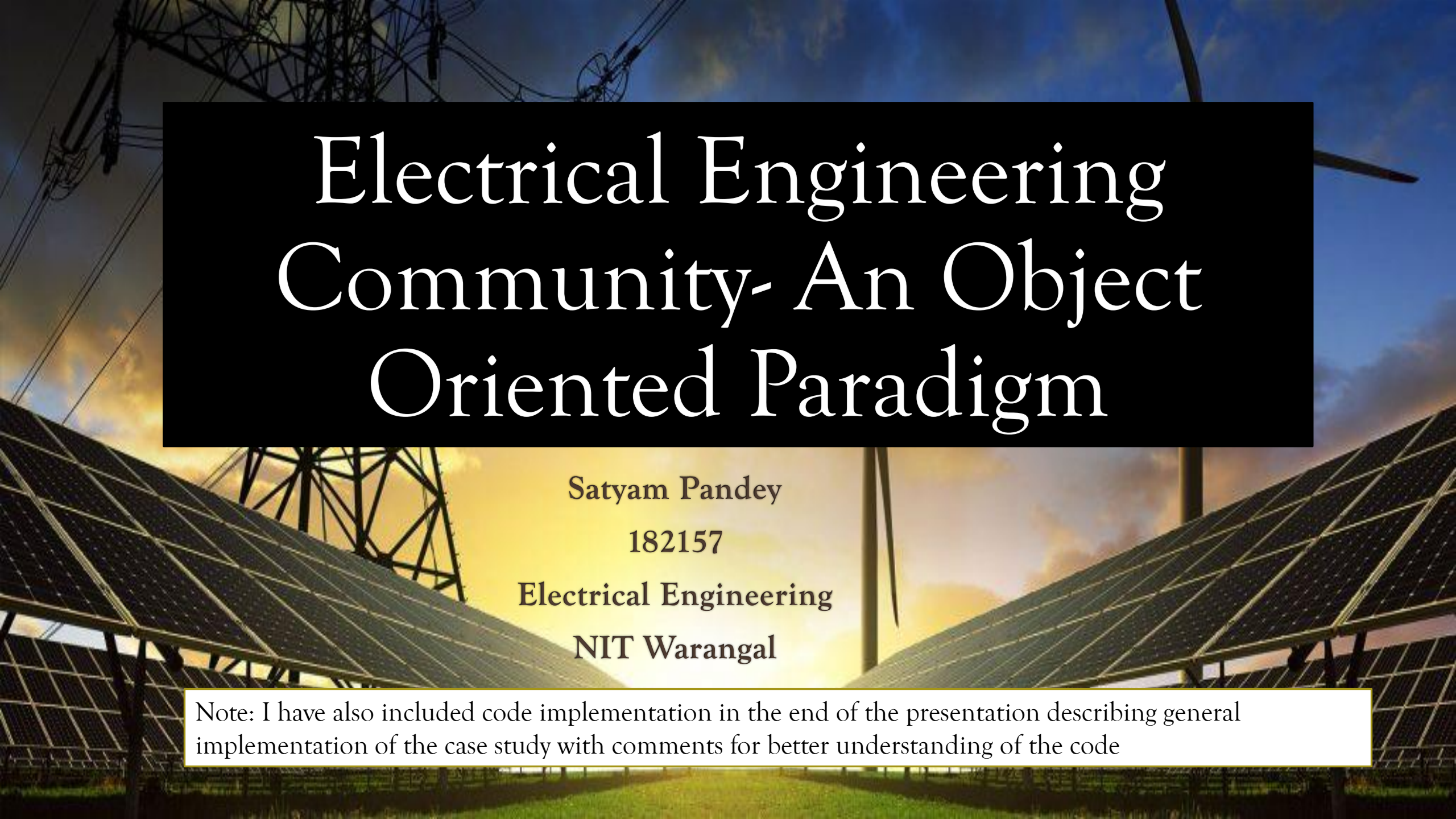


Electrical Engineering Community- An Object Oriented Paradigm

Satyam Pandey

182157

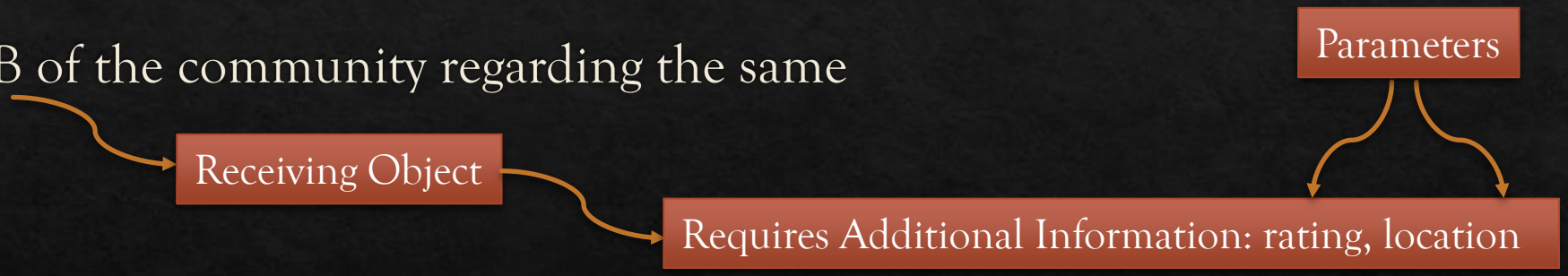
Electrical Engineering

NIT Warangal

Note: I have also included code implementation in the end of the presentation describing general implementation of the case study with comments for better understanding of the code

Messages and Methods

- Say, an object A needs a transformer to be installed in a village
- A communicates B of the community regarding the same



- B invokes method installation(rating, location), this method returns true if the asked rating say 25KVA is available in the stock and the location passed comes under their controlling region

- If installation function returns true, then B communicates the same to another object C in lower hierarchical level of Ministry of Power/Administration Hierarchy given authority for the issue
- C requires “clear reason” from B so as to why transformer needs to be replaced
- C invokes method approval(reason), if reason found satisfactory then installation is approved and the concerned engineer is given control thereafter
- Here reason may have predefined accepted values for reason like storm, relay failure etc. so that one can do it automatically by using switch statement, if reason found not of any predefined categories then it will be required for manual approval (we may trigger it by making use of default case in switch statement)

Class

- “Classes are not themselves the objects they describe”, say in the example which I cited, Junior Engineer is a class whose objects are given role to sort the issue of installation
- An object of this class depending on the given specification is open perform several functions like testing, handling, power management, etc. which belong to this class
- These objects have some states and behavior in common which make them part of same class

Inheritance

- Subclasses are not limited to the state and behaviors provided to them by their superclass
- Consider Ministry of Power as a hierarchical structure, allows descendants of a class to inherit all of its member elements and methods from its ancestors as well as creates its own
- In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class

Polymorphism

- As from the name it means “Many forms”
- Method Overriding: Functions with same name and same parameters and same return type, when derived class object calls the function, it calls overridden function in the derived class
- For method overloading, refer next slide

Method Overloading

- Methods with same name but different parameters
- Say, a village A is granted only issue of 15KVA transformer only to be installed at several places, no other transformer can be procured
- Here passing parameter of KVA rating is vague, so objects of transformer can be initialized(constructor) with 15 KVA
- Now both these constructors have same name but one is parameterized and other is not, so they fall under method overloading
- Overloading implements Compile Time Polymorphism

Abstraction

- Abstraction gives you the facility to define objects that represent abstract "actors" that can perform work, report on and change their state, and "communicate" with other objects in the system, like here communication between several objects of the electrical engineering community addressing a common problem of transformer procurement
- Here, the village people need not know the implementation details, they are just worried of installation, the process in between is immaterial to them, we are "hiding the implementation details from the user, only the functionality will be provided" to the user

Encapsulation

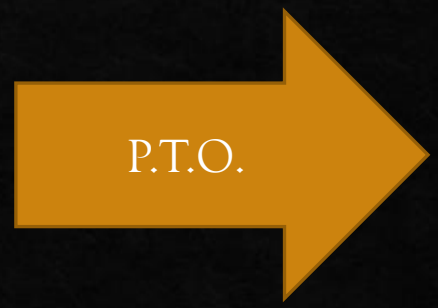
- A mechanism of wrapping the variables and code acting on the data (methods) together as a single unit that is putting the member elements and methods into together in the definition of a class is called encapsulation
- Specification of several class of machines/ministries/departments should be protected by disallowing/restricting access to some of the object’s components, only the object's own methods can directly inspect or manipulate its fields
- Hiding the internals of the object protects its integrity by preventing users from setting the internal data of the component into an invalid or inconsistent state, interf

Exceptions/Errors

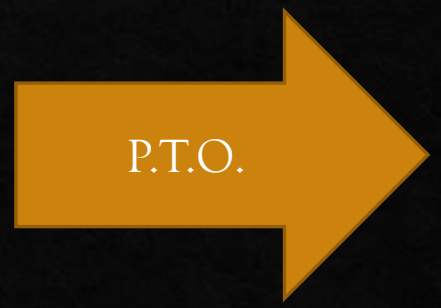
- Response to an exceptional circumstance
- Abnormal condition handling of relay, moving from normal condition (if(x<defined set value||x>defined set value), defining method for such operation condition
- The department in the lower level of the hierarchy fails to attend the request
- False claims
- Shortage of Equipment, Installation Issues, etc

Code Implementation (the code is long so spread over multiple slides)

```
/*code by Satyam Pandey*/
//I have provided comments whereever necessary to decipher my code, the code is executable
import java.util.*;
import java.lang.*;
import java.io.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
/* Here object of Location(village) Class will have a registered id which can be recognized by controlling station, the id will be specified by the user */
class Location {
    int identity;
    int rating;
    Location(int id, int rat) {
      identity=id;//Registered Id
      rating=rat;//in KVA
    }
```



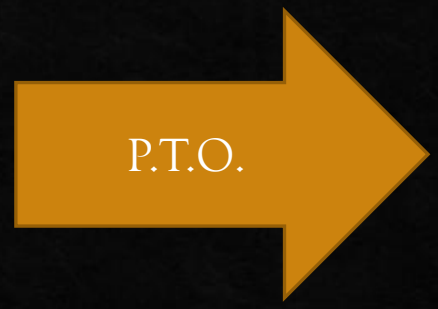
```
public static void main(String[] args) throws IOException, ParseException{
        BufferedReader inp = new BufferedReader(new InputStreamReader(System.in));
        //input reg id and rating of transformer
            int id= Integer.parseInt(inp.readLine());
            int rat= Integer.parseInt(inp.readLine());
        //for request to be valid here I am taking a condition for the sake of understanding that id>=1&&id<=20000, we have only 20000 villages currently registered
        System.out.println("Generate Request for Location ID: "+id+" for procurement of transformer of "+rat+" KVA rating");
        Location a = new Location(id,rat);
        String reason=inp.readLine();//to Input your reason for transformer amongst three categories storm, fault, relayfail, burnout only if anything else will be given it will not be accepted
        System.out.println("Given Reason: "+reason);
        TransProc b = new TransProc(id,rat,reason);// Creating object of Transformer Procurement authority class
        if(b.validregid()==true&&b.validreason()==true)
        {
            System.out.println("Your request for transformer procurement is Valid and shall be processed soon");
        }
```



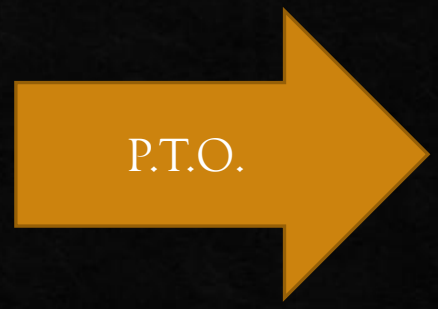
```
else
        {
            System.out.println("Sorry! Our team found your request as invalid and it will not be processed as of now");
            System.out.println("Now you need to seek approval from SDM, please enter the date when last time transformer was installed in your location: ");
            String datein=inp.readLine();//in dd/mm/yyyy format only
            Date datefor=new SimpleDateFormat("dd/MM/yyyy").parse(datein);
            System.out.println(datein+"\t"+datefor);  //in GMT



            AdminApprov c= new AdminApprov(datein);
            if(c.Canapprove()==true)
            {
                System.out.println("Yes, Ministry has approved your request because it was installed last time quite back, so your request will be processed now!!");
            }
            else
            System.out.println("Sorry! Your request cant be processed further as it is also rejected by ministry since it was installed within 2 years back only");


        }

    }
}
```

P.T.O.

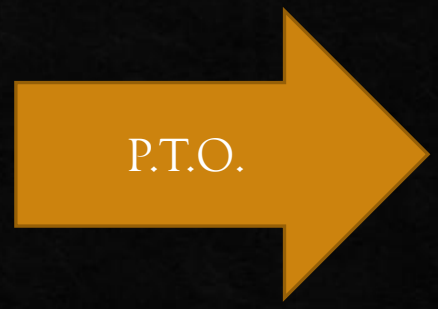
```
class TransProc {
    int regid;
    int reqrating;
    String reason;
   TransProc(int id, int reqrat, String reas) {
       regid=id;
       reqrating=reqrat;
       reason=reas;
      System.out.println("Request for Transformer Procurement Received, Controlling Authority to attend the request for the following specifications:\n"+"Location Reg ID: "+regid+"\n"+"Transformer Rating in KVA: "+reqrating+"\n"+"Reason given by Location Incharge: "+reason);
   }
   boolean validregid()
   {

       if(regid>=1&&regid<=20000)
       {
           return true;
       }
       else
       return false;

   }
```



```
boolean validreason()
   {


if(reason.equals("storm")||reason.equals("relayfail")||reason.equals("burnout")||reason.equal
s("fault")||reason.equals("naturalhazard"))
        return true;
        else
        return false;
   }


}
```



```
class AdminApprov {
    String start;
    String end;
    Date datestart;
    Date dateend;
    String dateinn;
    Date datein1;
    AdminApprov(String datein) throws ParseException
    {
        start="11/10/2007";
         end="11/10/2018";
        datestart=new SimpleDateFormat("dd/MM/yyyy").parse(start);
        dateend=new SimpleDateFormat("dd/MM/yyyy").parse(end);

        dateinn=datein;
        datein1=new SimpleDateFormat("dd/MM/yyyy").parse(dateinn);
    }
```



```
boolean Canapprove()
    {
        if(datein1.after(datestart) && datein1.before(dateend))
        return true;
        else
        return false;
    }


}
//end of program
```

Thank You