ECE 651
Lecture 1: Design Principles
Notes Outline

- **Think/Pair/Share** What do customers want (in general)?
  - What characteristics of software?
  - Which are in opposition (having one makes the other hard)?Why
  - Which are synergistic (having one makes the other easier)?Why
- Three big goals in opposition to each other:
- Challenges in parallelizing development tasks:
- How can we be successful at parallelizing development tasks?

- OO Review from 551:
  - Inheritance:
  - Polymorphism:
  - Abstraction:
  - Encapsulation:
- Least Surprise
- Don't Repeat Yourself (DRY)
- Low Coupling/High Cohesion
- Single Responsibility Principle(SRP)

- Example SRP violation code:

```
  DataSet readAndParse() {
   ConfigFileReader cfr = new ConfigFileReader(\config.txt");
   String addr= cfr.getLineFor(\datasource").getValue();
   InternetAddress addr = new InternetAddress(addr);
   Socket sock = new Socket(source);
   InputStream inp = sock.getInputStream();
   String line;
   DataSet ans = new DataSet();
   while((line = inp.readLine()) != null){
       String[] parts = splitUp(line);
       //some error checking on first part
       firstPart = doSomething(parts[0]);
       //some error checking on second part
       secondPart = doOtherThing(parts[1]);
       //some error checking on third part
       thirdPart = anotherFn(parts[2]);
       ans.add(new Data(firstPart, secondPart, thirdPart));
   }
   return ans;
}
```

- SRP violation at class level:

```
class DataParser {
   public:
      InternetAddress getSourceAddress() {...}
      InputStream getInputStream(InternetAddress addr) {...}
      DataSet parseData(InputStream inp) {...}

}
```

- Open/Closed Principle

- Open/Closed violation code:

```
void someMethod(Bird b) {
   if (b.getType() == Bird.PIGEON) {
      //pigeon code
   }
   else if (b.getType() == Bird.PARROT) {
       //parrot code
   }
   else if (b.getType() == Bird.EAGLE) {
       //eagle code
   }
   //...
}
```

- What if we add Penguin to our code?

- Penguin violates postconditions promised by Bird

```
class Bird {
   //postcondition: bird will be flying towards altitude
   virtual void fly(int altitude) {
          //whatever code to make it fly up to height
    }
};
class Penguin : public Bird
   //postcondition: bird's behavior unchanged
   virtual void fly(int altitude) {
        //do nothing: penguins dont fly
   }
};
```

- Liskov Substitution Principle: Big idea

- Preconditions and postconditions example
  - Linked List:

    ```
    template<typename T>
    class LinkedList {
       //precondition:  0<=index< number of elements in list
       //postcondition: return value is indexth element
       T& operator[] (size_t index) {...}

       //precondition: none
       //postcondition: this list contains no values that == toRemove
       void removeAll(const T& toRemove) {...}
    };
    ```

  - Code that uses LinkedList:

    ```
      void myFunction(LinkedList & ll) {
       for (size_t i = 0; i < ll.getSize(); i++) {
    //we expect this to work:
    //precondition is 0<=index<size, and we enforce that
             T& data = ll[i];
           //other code that uses data
        }
    }

    void otherFn(LinkedList & ll){
        //some other code
        ll.removeAll(someValue);
        //you expect ll not to have someValue at all here
        //and this code might rely on it
    }
    ```

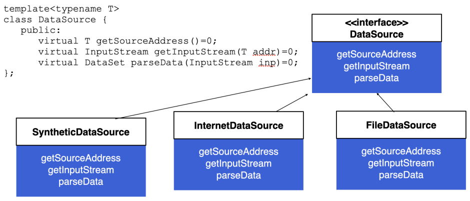
- Liskov Substitution Principle:
  - Preconditions:

  - Postconditions:

  - Invariants:

  - History:

- Interface Segregation Principle:



```
template<typename T>
class DataSource {
   public:
      virtual T getSourceAddress()=0;
      virtual InputStream getInputStream(T addr)=0;
      virtual DataSet parseData(InputStream inp)=0;
};
```

- Dependency Inversion Principle:

- Dependency Injection:

```
class Manager  {
    std::vector<Developer> developers;
     void addDeveloper() {
               developers.push_back(Developer()); //not dependency injection
    }

}
```

```
class Manager  {
    std::vector<Employee *> employees;
     void addEmployee(Employee * e) {  //dependency injection
                  employees.push_back(e);
    }

}
```

- Designing for Testing

- How does SRP help with testing?