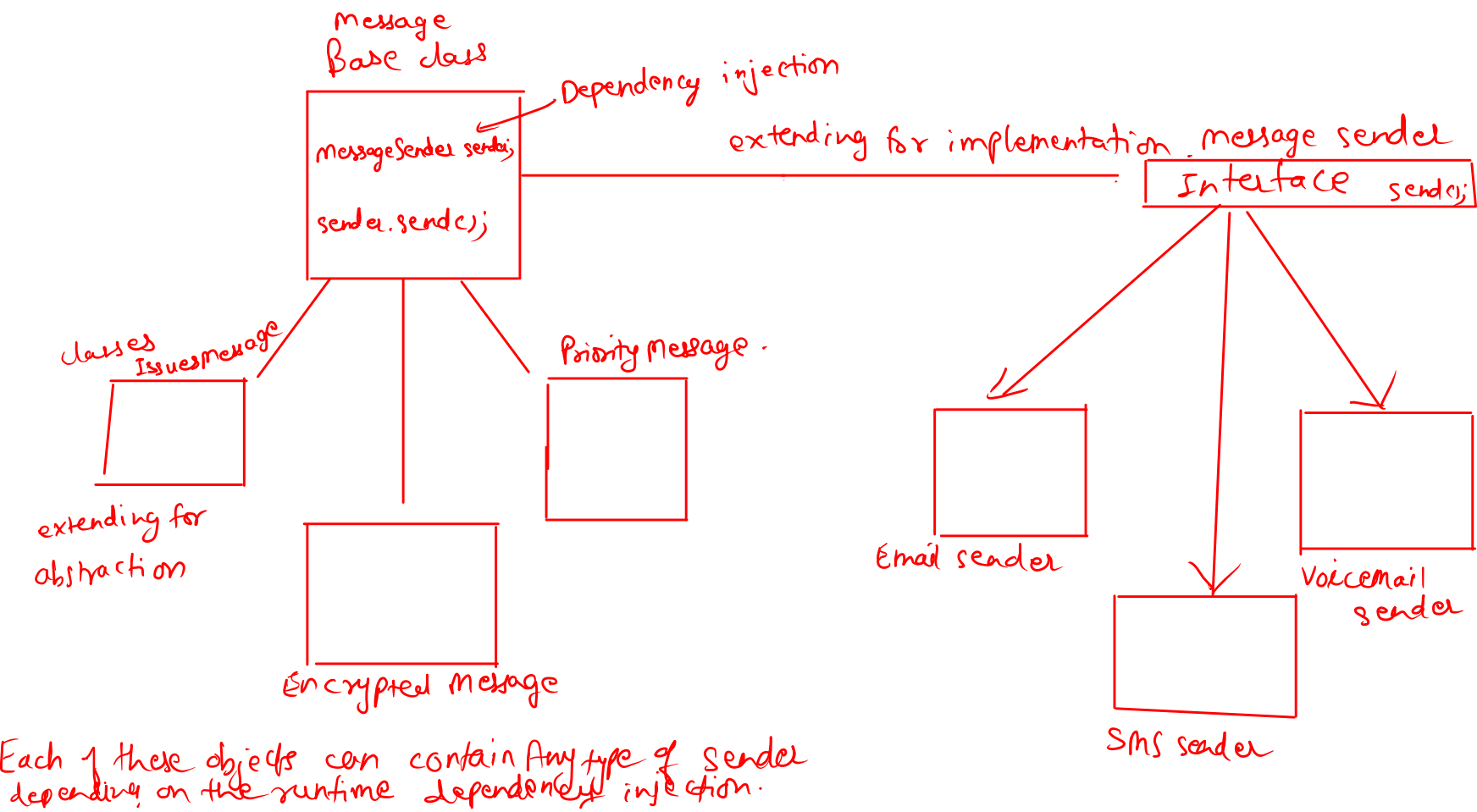
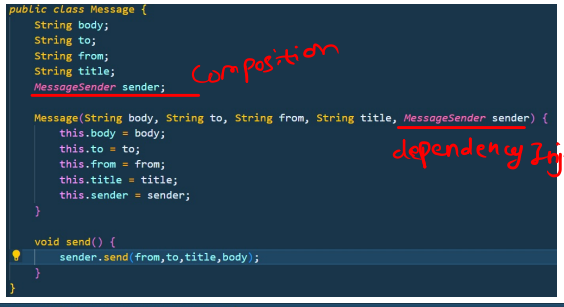
Bridge Pattern

There can be 2 reasons we extend a class, to change the level of abstraction or to change the implementation of the functions written inside it.

In Bridge Pattern we compose an interface reference in the base class for the classes which are ment to extend the base class for defining different implementation of it.

This in turn saves the Bursting of classes and we can achive loose coupling with these kinds of classes.


Message
Base class
Dependency injection
MessageSender sender;
sender.send();
extending for implementation
message sender
Interface
sender;
classes
IssueMessage
Priority Message.
extending for
abstraction
Encrypted message
Email sender
Voicemail
sender
SMS sender
Each of these objects can contain Any type of sender
depending on the runtime dependency injection.




```java
public class Message {
    String body;
    String to;
    String from;
    String title;
    MessageSender sender;

    Message(String body, String to, String from, String title, MessageSender sender) {
        this.body = body;
        this.to = to;
        this.from = from;
        this.title = title;
        this.sender = sender;
    }

    void send() {
        sender.send(from,to,title,body);
    }
}
```

```java
public interface MessageSender {
    void send(String from,String to,String title, String body);
}
```

(abstraction classes)

(implementation classes)

```java
public class PriorityMessage extends Message {
    int priority;

    PriorityMessage(String body, String to, String from, String title, int priority, MessageSender sender) {
        super(body, to, from, title, sender);
        this.priority = priority;

        if(priority > 50) {
            body += "important message 50";
        } else if(priority > 60) {
            body += "important message 60";
        }
    }
}
```

```java
public class SMSSender implements MessageSender {
    @Override
    public void send(String from, String to, String title, String body) {
        System.out.println("----------------------------------------");
        System.out.println("Title: " + title);
        System.out.println("Body: " + body);
        System.out.println("from: " + from);
        System.out.println("To: " + to);

        System.out.println("Send via Email: ");
    }
}
```

```java
public class EncryptedMessage extends Message {
    String key;
    EncryptedMessage(String body, String to, String from,String title, String key, MessageSender sender) {
        super(body, to, from, title,sender);
        this.key = key;
        this.body = key + body;
    }
}
```

```java
public class EmailSender implements MessageSender {
    @Override
    public void send(String from, String to, String title, String body) {
        System.out.println("----------------------------------------");
        System.out.println("Title: " + title);
        System.out.println("Body: " + body);
        System.out.println("from: " + from);
        System.out.println("To: " + to);

        System.out.println("Send via Email: ");
    }
}
```

```java
public class IssuesMessage extends Message {
    ArrayList<String> issues;

    IssuesMessage(String body, String to, String from,String title, ArrayList<String> issues, MessageSender sender) {
        super(body, to, from, title,sender);
        this.issues = issues;

        body+= "Following issues:-  ";
        for(String issue: issues) {
            body += issue;
        }
    }
}
```

```java
public class VoiceMailSender implements MessageSender {
    @Override
    public void send(String from, String to, String title, String body) {
        System.out.println("----------------------------------------");
        System.out.println("Title: " + title);
        System.out.println("Body: " + body);
        System.out.println("from: " + from);
        System.out.println("To: " + to);

        System.out.println("Send via VoiceMail: ");
    }
}
```

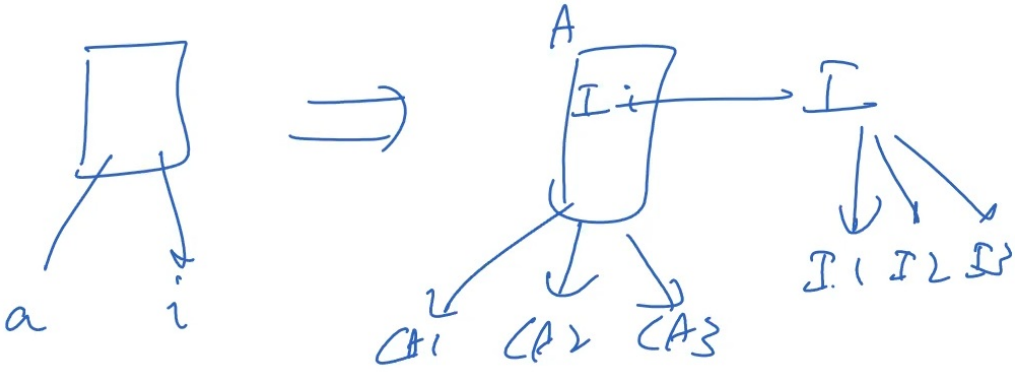
```java
class Test {
    Run | Debug
    public static void main(String[] args) {
        VoiceMailSender voiceMail = new VoiceMailSender();
        EncryptedMessage encrytpted = new EncryptedMessage(body: "ecnrypt", to: "sad", from: "sda",
        title: "ss", key: "56", voiceMail);
        encrytpted.send();

        SMSSender smsSend = new SMSSender();
        PriorityMessage priority = new PriorityMessage(body: "bosy", to: "to", from: "from",
        title: "tit;e", priority: 56, smsSend);
        priority.send();

        VoiceMailSender vcMail = new VoiceMailSender();
        ArrayList<String> issues = new ArrayList<>();
        issues.add(e: "issue 1"); issues.add(e: "issue 2"); issues.add(e: "issue 3");
        IssuesMessage issuesMessage = new IssuesMessage(body: "Issues", to: "sarKar", from: "me",
        title: "ISSUES", issues,vcMail);
        issuesMessage.send();
    }
}
```

```
----------------------------------------
Title: ss
Body: 56ecnrypt
from: sda
To: sad
Send via VoiceMail:
----------------------------------------
Title: tit;e
Body: bosy
from: from
To: to
Send via Email:
----------------------------------------
Title: ISSUES
Body: Issues
from: me
To: sarKar
Send via VoiceMail:
```

Cohesion just means readability of the code.



10 A x 5 I → 50 classes

Bridge → (10 A, 5 I + 2)

no need of 50 classes if we have 10 Abstraction & 5 Implementation classes, we only need 17 classes 10 Abstraction, 5 Implementation 1 Base class including 1 interface for those 5 Implementation classes.