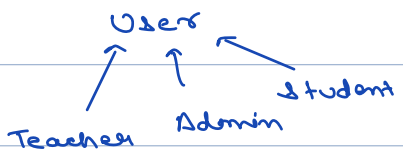
× not

Agenda :- Simple factory
factory method
Abstract factory.

creational

```java
class Student extends User {
    public Student(String firstName, String lastName) {
        super(firstName, lastName);
    }
    // Student-specific methods
}

class Teacher extends User {
    public Teacher(String firstName, String lastName) {
        super(firstName, lastName);
    }
    // Teacher-specific methods
}

class Admin extends User {
    public Admin(String firstName, String lastName) {
        super(firstName, lastName);
    }
    // Admin-specific methods
}
```




```java
// In the client code
public class Client {
    public static void main(String[] args) {
        User user;
        String userType = getUserTypeFromInput(); // This could be from user input,

        if (userType.equals("student")) {
            user = new Student("John", "Doe");
        } else if (userType.equals("teacher")) {
            user = new Teacher("John", "Doe");
        } else if (userType.equals("admin")) {
            user = new Admin("John", "Doe");
        } else {
            throw new IllegalArgumentException("Invalid user type");
        }

        // Use the created user object
    }
}
```

API, bg job.

Object creation could be complex.

1. Coupling two classes
2. OCP principle — we are breaking these..
3. Duplication → is hard to update — if want to add more then.. we will have to add more if else condn.
4. SRP principle — Object creation may be tough.. this may call some api , db.

① Simple factory.

```
class UserFactory {
  public static User createUser(UserRole role) {
    switch (role) {
      case STUDENT:
        return new Student("John", "Doe");
    case TEACHER:
      return new Teacher("John", "Doe");
    case ADMIN:
      return new Admin("John", "Doe");
    }
  }
}
```

I/P

Enum
↳ — Student,
— . Teacher
— . Admin.
—

client

↳ User user = UserFactory.createUser
(userType. —) ;

I/O

WebService {

Database db = ——— (mysql, mongodb, fakeDB)

createWeb() {

if (db is inst of MySQL) {
| Query q = mySQLQuery();
else if (db is mongodb) {
| Query q = mongodbQuery();
}
.
.
.

db.execute(q);
}

class QueryFactory {

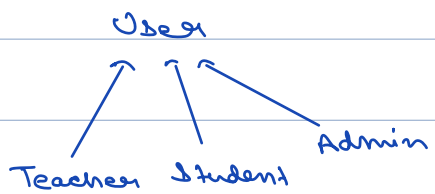
——— (List v) {

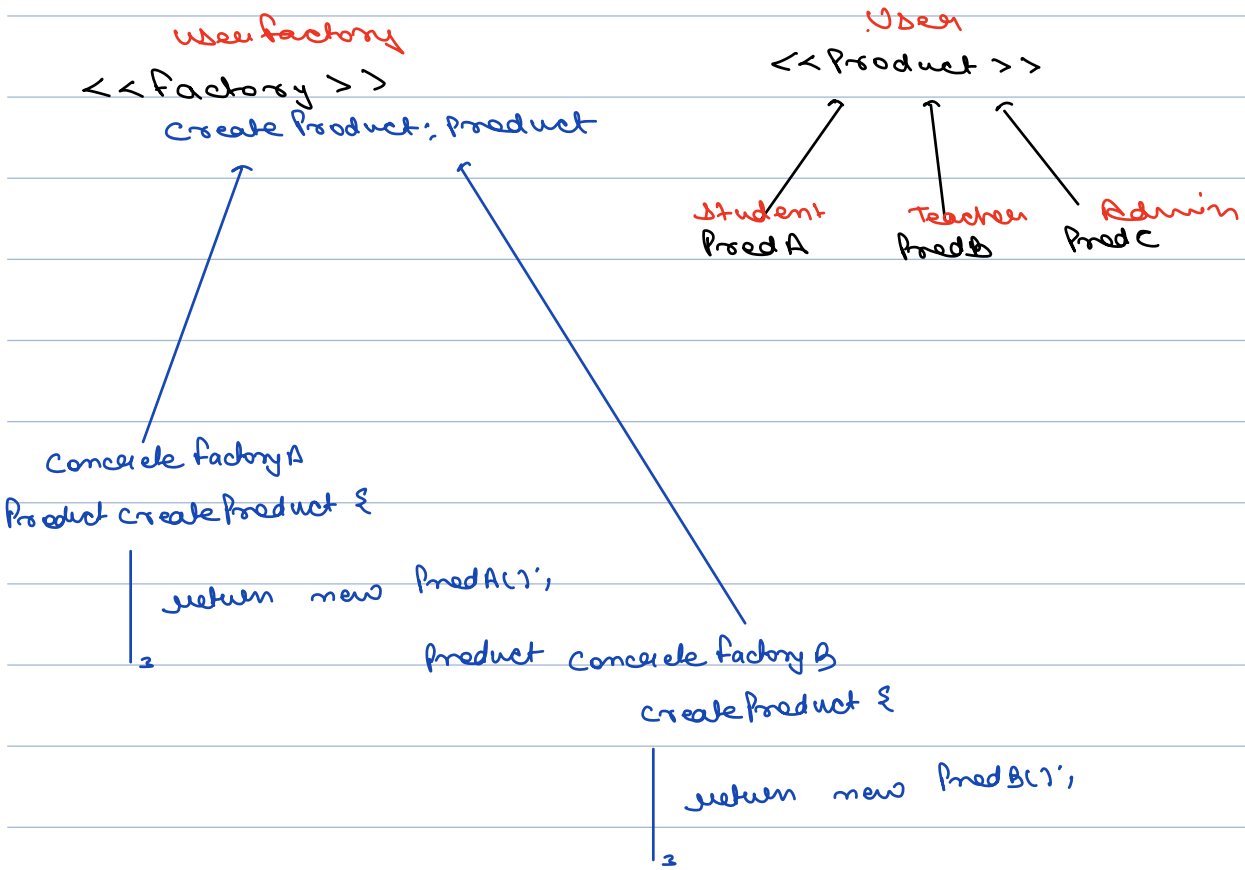
for —
switch (mysql) {
return mySQLQuery();
}
}

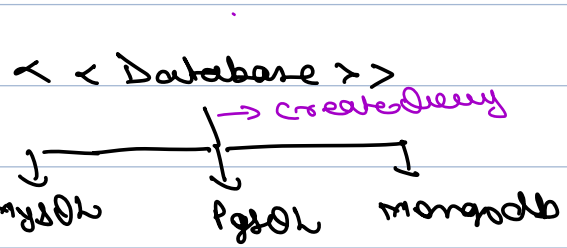
2) Factory method pattern:-

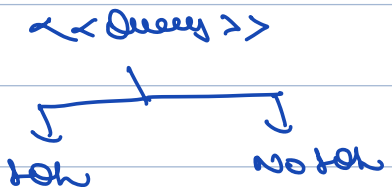
```
class UserFactory {
  public static User createUser(UserRole role) {
    switch (role) {
      case STUDENT:
        return new Student("John", "Doe");
    case TEACHER:
      return new Teacher("John", "Doe");
    case ADMIN:
      return new Admin("John", "Doe");
    }
  }
}
```

OCP

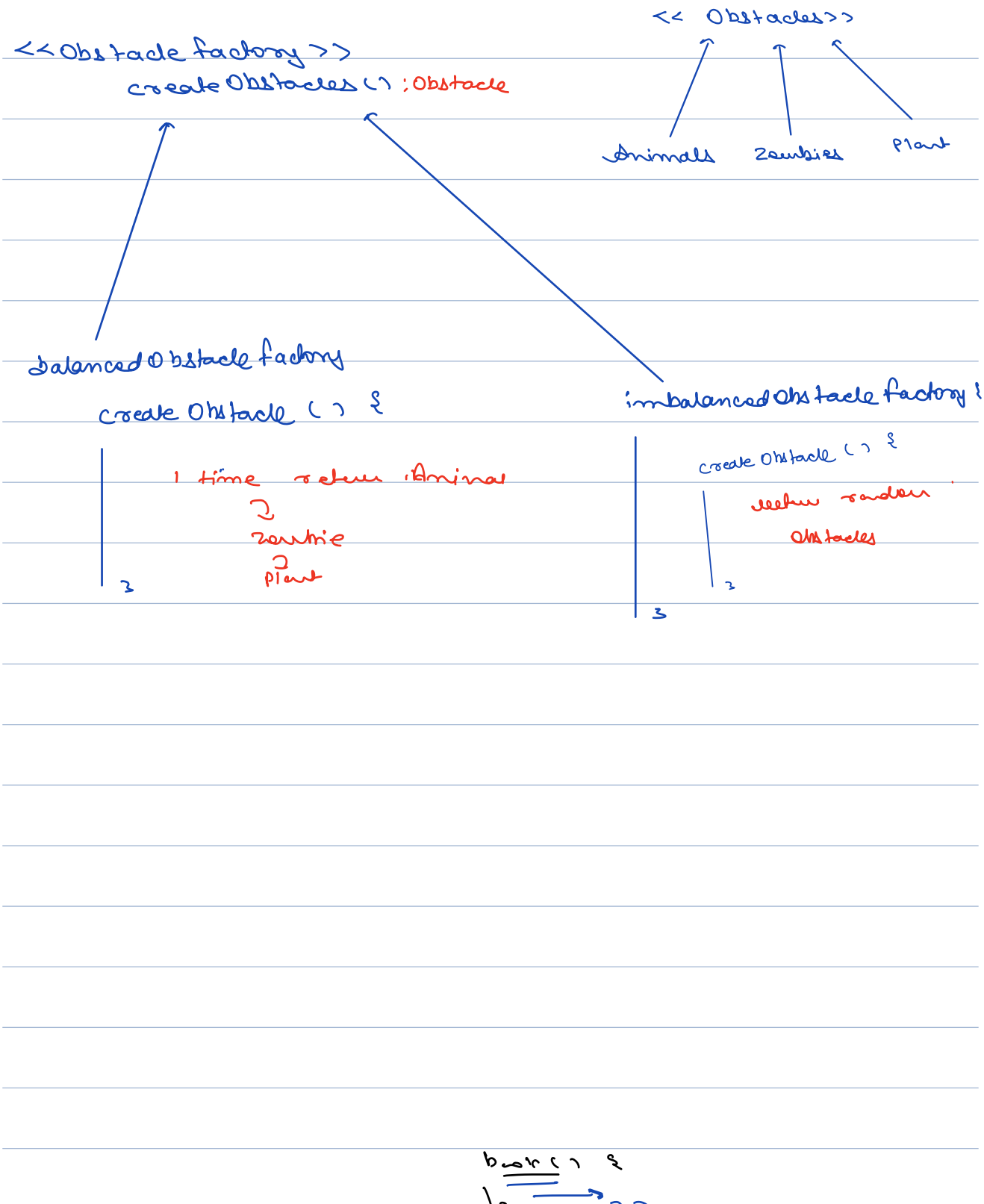



User factory
<<Factory>>
Create Product: Product
User
<<Product>>
Student
Prod A
Teacher
Prod B
Admin
Prod C
Concrete factory A
Product createProduct {
return new ProdA();
}
Product Concrete factory B
createProduct {
return new ProdB();
}

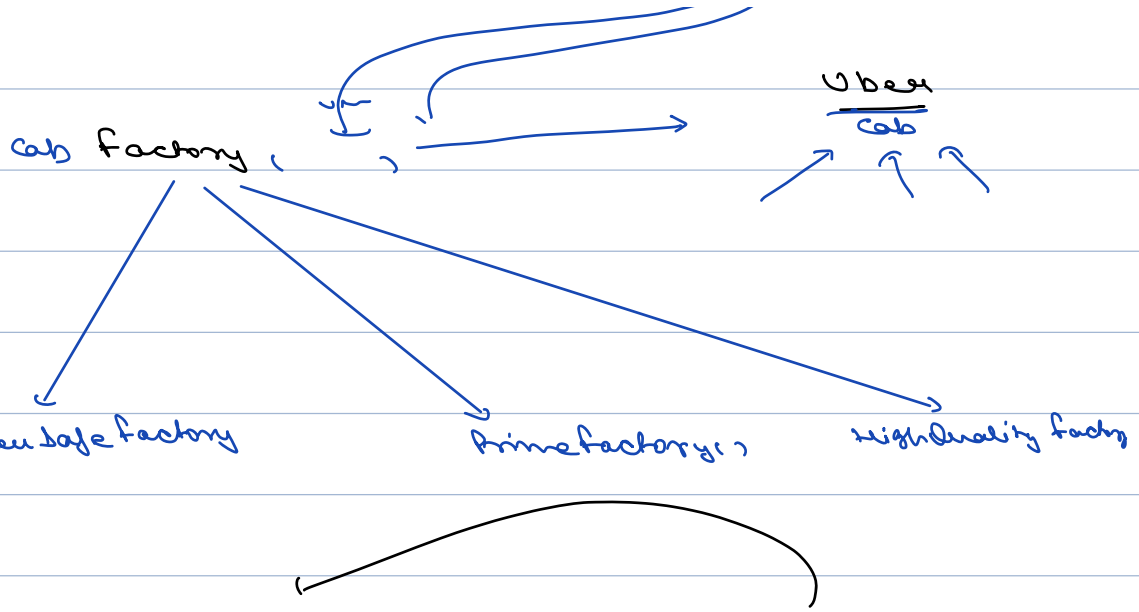

<<Database>>
createQuery
mySQL
PgSQL
mongodb


<<Query>>
SQL
NoSQL

Video Game



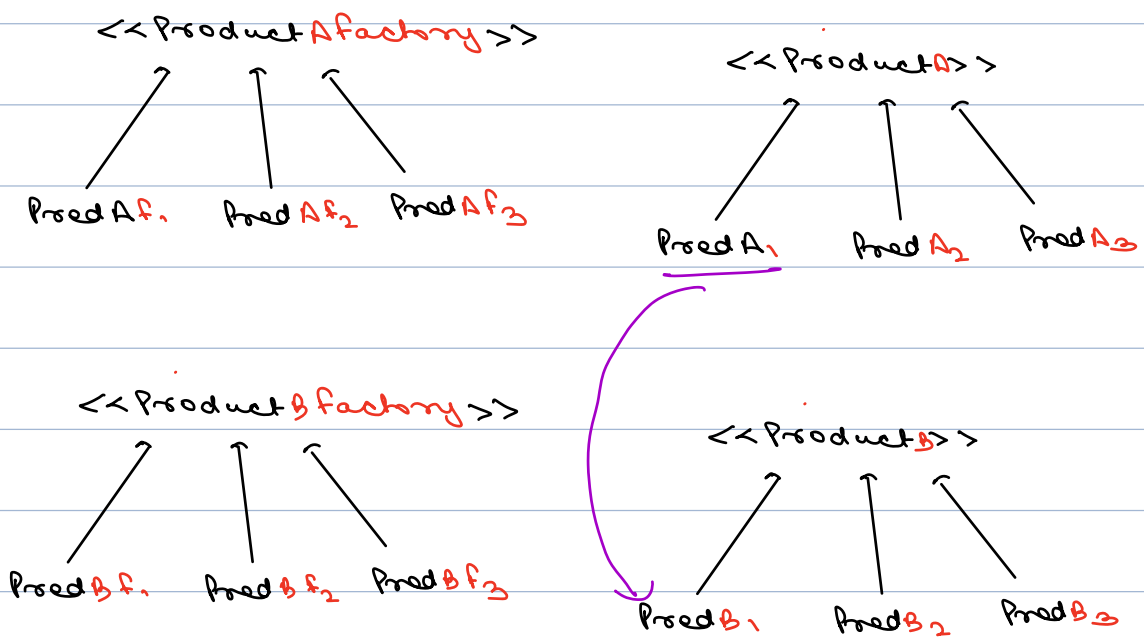
boom () {




```
Cab c = CabFactory.getCab(User, Time);
```

⑤ Abstract factory

factory of factories



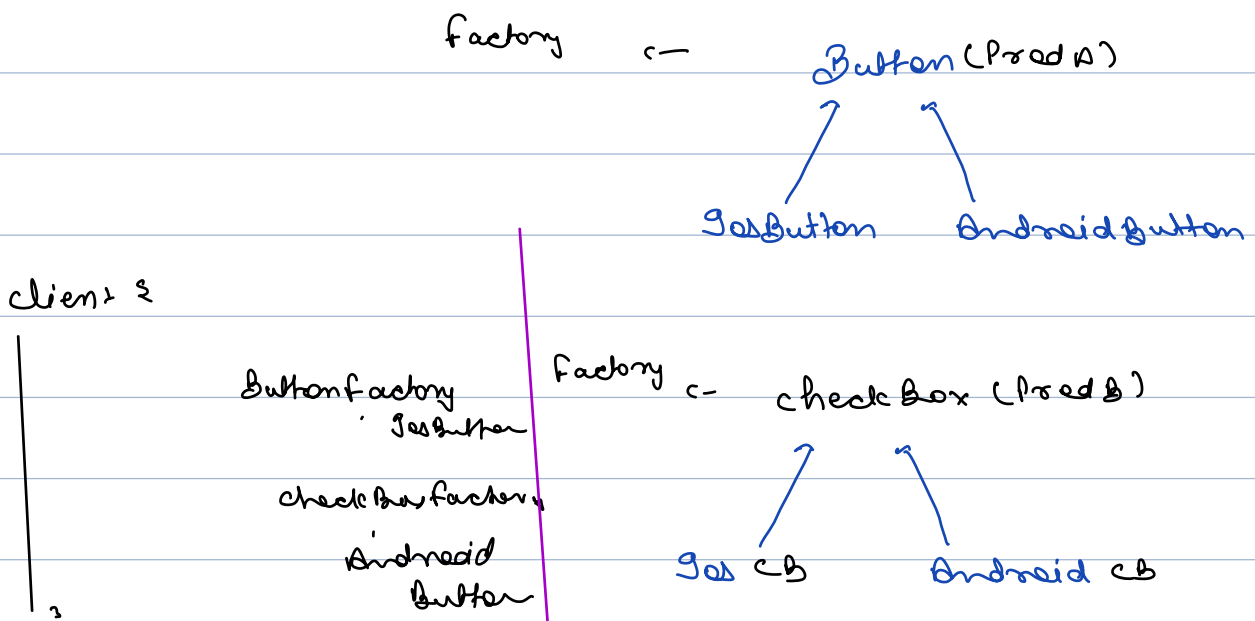
client {

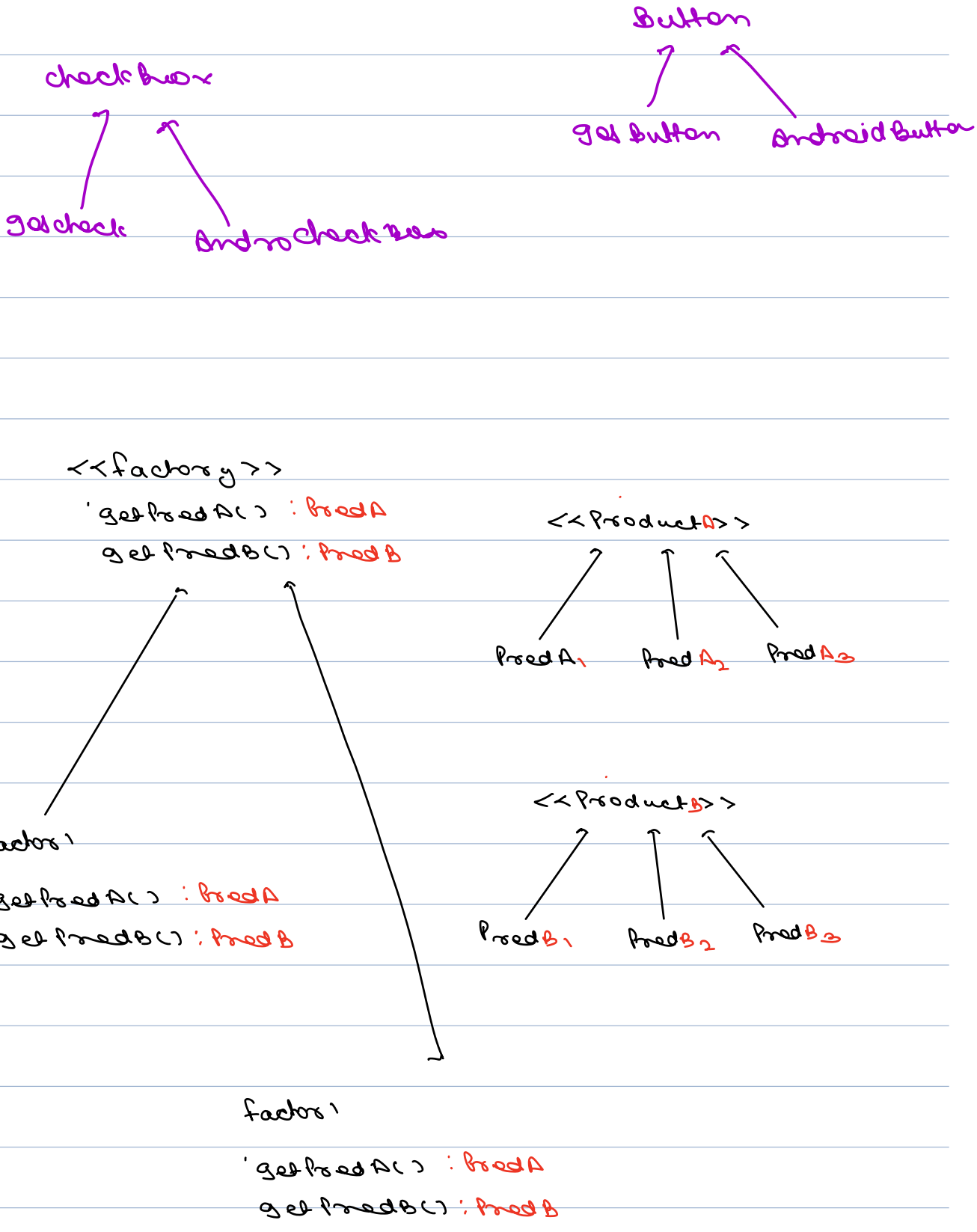
ProdA, ProdB

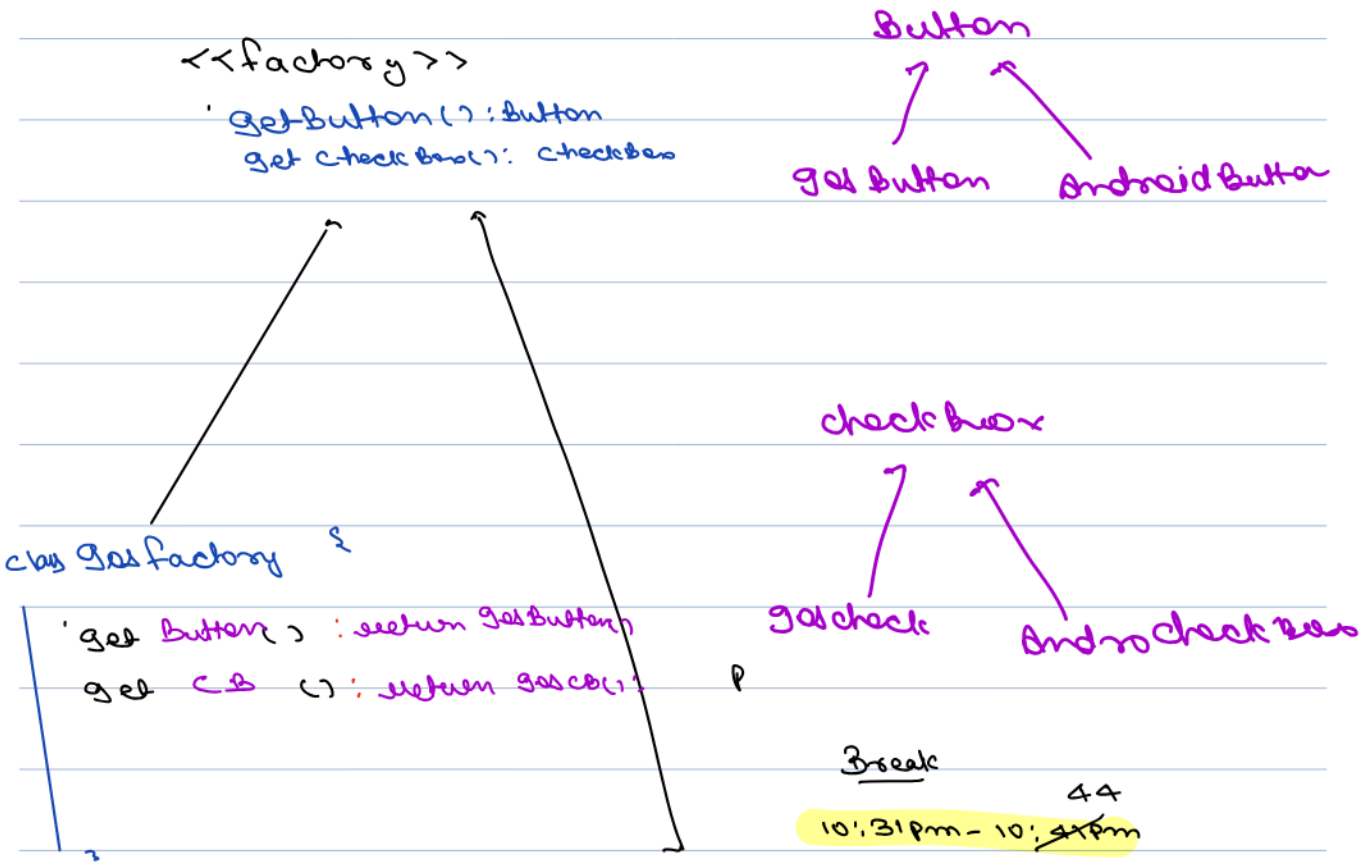
→ ProductAfactory

→ ProductBfactory

}




Button
GetButton
AndroidButton
CheckBox
GetCheck
AndroCheckBox
<<factory>>
'getProdA() : ProdA
getProdB() : ProdB
<<ProductA>>
ProdA1
ProdA2
ProdA3
<<ProductB>>
ProdB1
ProdB2
ProdB3
factory1
'getProdA() : ProdA
getProdB() : ProdB
factory1
'getProdA() : ProdA
getProdB() : ProdB



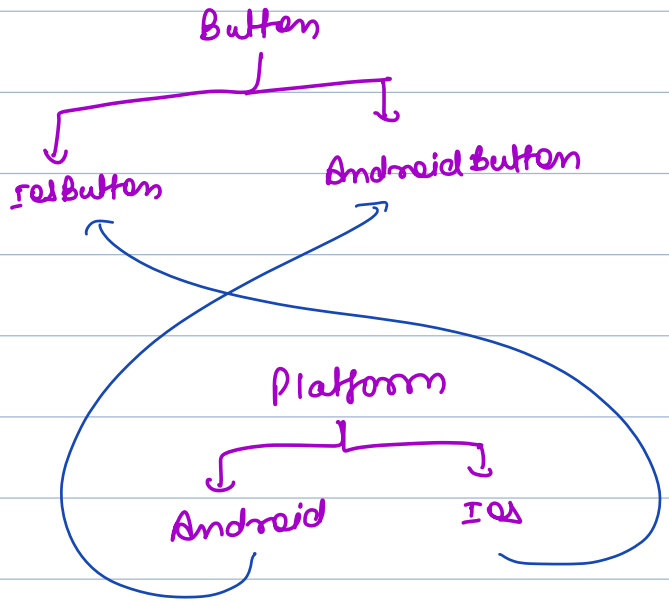

```
class StudentFactory {
  public User createStudent(String firstName, String lastName) {
    return new Student(firstName, lastName);
  }
}
class TeacherFactory {
  public User createTeacher(String firstName, String lastName) {
    return new Teacher(firstName, lastName);
  }
}
```

```
abstract class ClassroomFactory {
  public abstract Student createStudent(String firstName, String lastName);
  public abstract Teacher createTeacher(String firstName, String lastName);
}
```

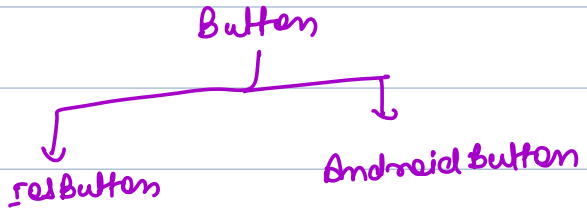
```
 class BiologyClassroomFactory extends ClassroomFactory {
   @Override
   public Student createStudent(String firstName, String lastName) {
     return new BiologyStudent(firstName, lastName);
   }

   @Override
   public Teacher createTeacher(String firstName, String lastName) {
     return new BiologyTeacher(firstName, lastName);
   }
 }
```

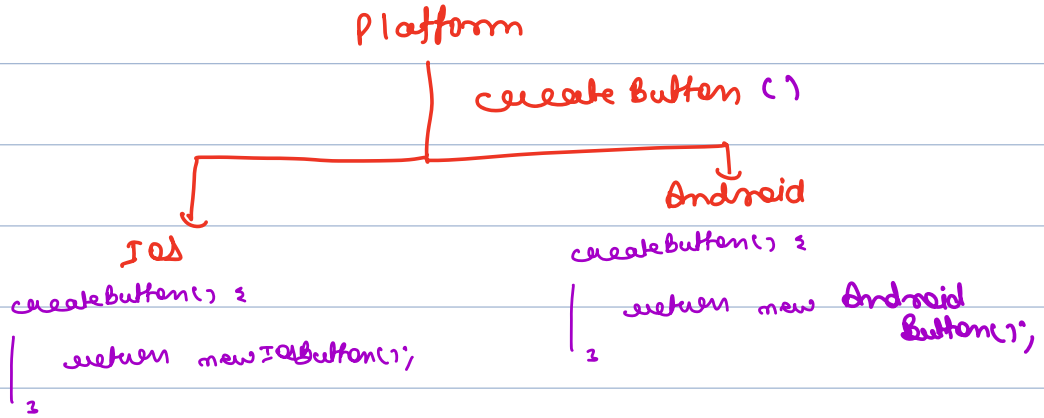
```
ClassroomFactory factory = new BiologyClassroomFactory();
Student student = factory.createStudent("John", "Doe");
Teacher teacher = factory.createTeacher("John", "Doe");
```

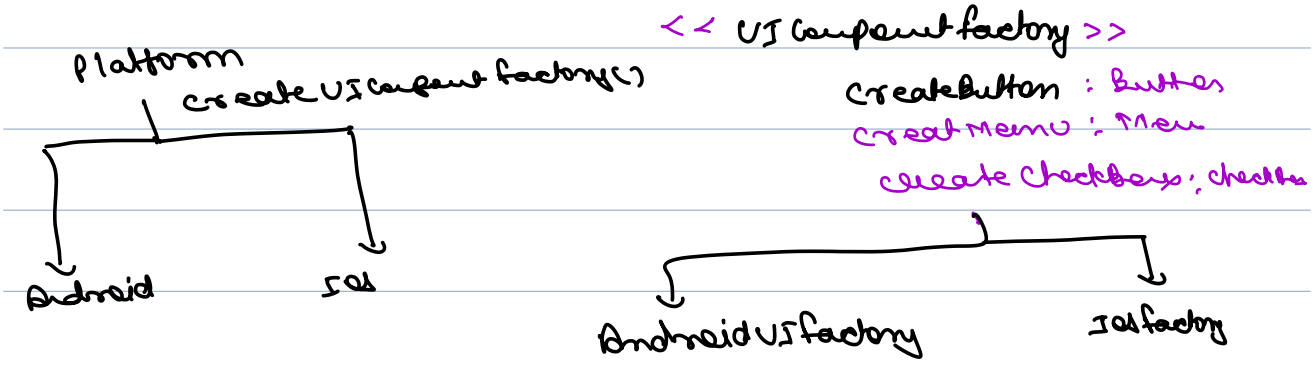
Flutter / React Native : cross platform UI.

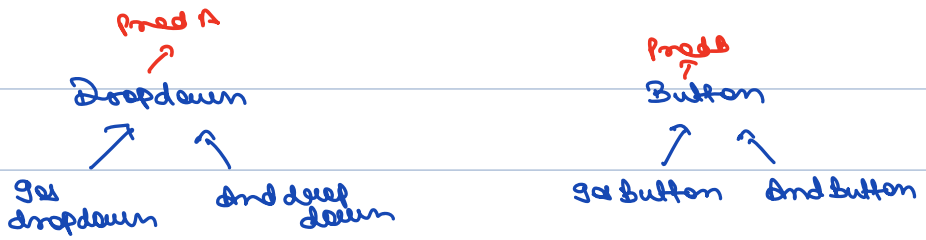


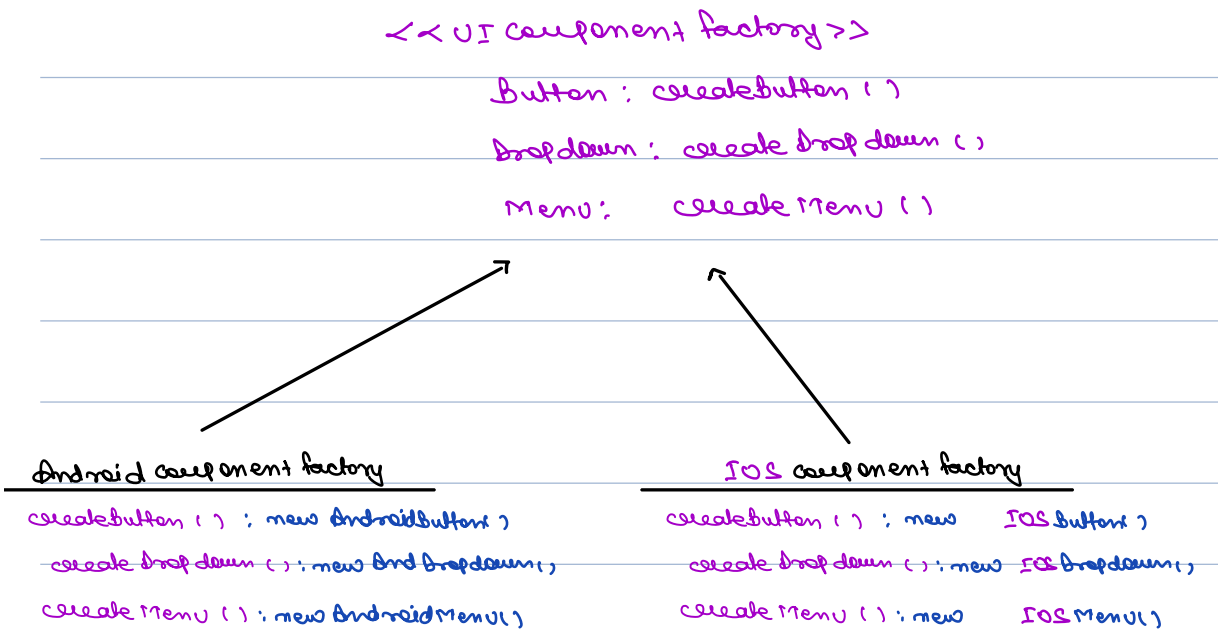

```
createButton() {
    if (Platform == IOS) {
        Button b = new IOSButton();
    } else {
        ——
    }
}
```

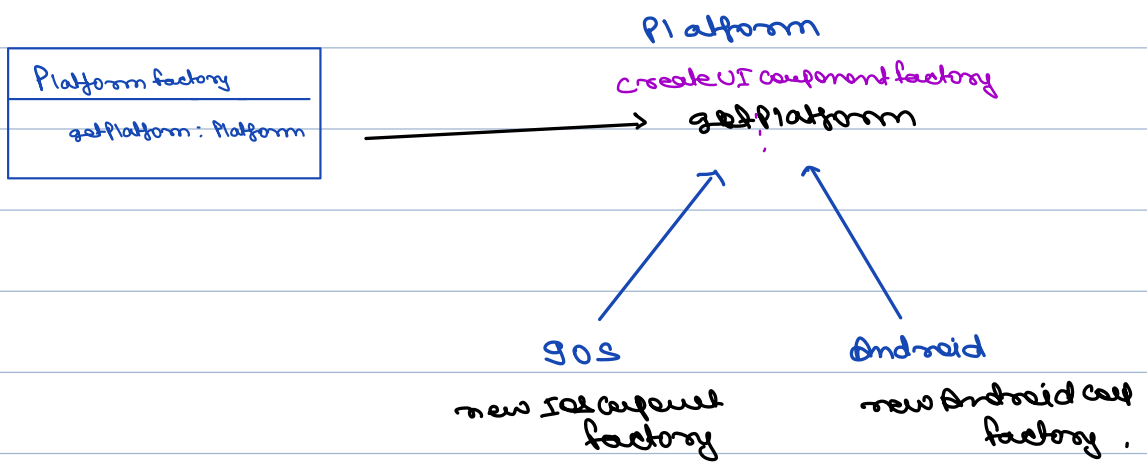

Button
IOSButton
AndroidButton


Platform
createButton()
IOS
createButton() {
return new IOSButton();
}
Android
createButton() {
return new AndroidButton();
}


Platform
createUIComponentFactory()
Android
IOS
<< UIComponentFactory >>
createButton : Button
createMenu : Menu
createCheckbox : Checkbox
AndroidUIFactory
IOSFactory


Prod A
Dropdown
IOS dropdown
And drop down
Prod B
Button
IOS Button
And Button


<< UI component factory >>
Button : createButton ( )
Dropdown : create Dropdown ( )
Menu : create Menu ( )
Android component factory
createButton ( ) : new AndroidButton ( )
create Dropdown ( ) : new AndDropdown ( )
create Menu ( ) : new AndroidMenu ( )
IOS component factory
createButton ( ) : new IOSButton ( )
create Dropdown ( ) : new IOSDropdown ( )
create Menu ( ) : new IOSMenu ( )


Platform factory
getPlatform : Platform
Platform
createUI component factory
getPlatform
IOS
new IOS component factory
Android
new Android component factory