Exercise 1

```java
public class Student {

  private String id;

  private University university;

  private int age;

  private List<Course>
registeredCourses;

  ...

}
```


```java
public class AlcoholRegulations {

  public boolean canLegallyDrinkAlcohol(Student st){

    switch (st.getUniversity().getCountry()) {

        case "Afghanistan":

              return false;

        case "Armenia":

              return true;

        case "Canada.BC":

              return (st.getAge >= 19);

        case "Canada.AB":

              return (st.getAge >= 18);

        ...

    }

  }

}
```

TODO

Refactor these classes to reduce their coupling.

Exercise 1

```
public class AlcoholRegulations {
   public boolean
canLegallyDrinkAlcohol(Student st){
     switch (st.getUniversity().getCountry()) {
          case "Afghanistan":
           return false;
          case "Armenia":
           return true;
          case "Canada.BC":
           return (st.getAge >= 19);
```

P2

P1

```java
public class Student {
    private String id;
    private University university;
    private int age;
    private List<Course> registeredcourses;
    private Country country;        //added
}
```

Unnecessary duplication of data

Still tied to the Student class

```java
public class AlcoholRegulations {
    public boolean canLegallyDrinkAlochol(Student st) {
        //here we removed the call to getUniversity()
        switch(st.getCountry()) {
            case "Afghanistan":
                return false;

        }
    }
}
```

ALTERNATIVE 1

No more coupling right now (only one class) :)
Bad cohesion :(

```
public class Student {
    private int age;
    private String id;
    private University university;
    private List<Course> registeredCourses;

    public boolean canLegallyDrinkAlcohol() {
            switch (this.getCountry()) {
                case "Afghanistan":
                    return false;
                case "Armenia":
                    return true;
                case "Canada.BC":
                    return (this.getAge() >= 19);
                case "Canada.AB":
                    return (this.getAge() >= 18);
            }

    public String getCountry() {
        return university.getCountry();
    }

    public int getAge() {
        return age;
    }
}
```

Still tied to the Student class

ALTERNATIVE 2

```
public interface Person {
    public int getAge();
    public University getUniversity();
}

public class Student implements Person {
    private String id;
    private University university;
    private int age;
    private List<Course> registeredCourses;

    public int getAge() {...}
    public University getUnivesity() {...}
}

public class AlcoholRegulation {
    public boolean canLegallyDrinkAlcohol (Person p) {
        switch (p.getUniversity().getCountry()) {
            case "Afghanistan":
                return false;
            case "Armenia":
                return true;
            case "Canada.BC":
                return (p.getAge()>=19);
            case "Canada.AB":
                return (p.getAge()>=18);
            ...
        }
    }
}
```

Limited to people that have a link with a university

Still some coupling
(AlcoholRegulation depends on Person)
More complex design :(

Demeter's principle still violated

ALTERNATIVE 3

```
public class Person{
        private String id;
        private int age;
        private String country;
    }

    public class Student extends Person{
        private String sid;
        private University university;
        private List<Course> registeredCourse;
        ...
    }

    public class AlcoholRegulation {
        public boolean canLegallyDrinkAlcohol(Person p){
            switch (p.getCountry()){
            case ...
            case "Canada.BC":
                return (p.getAge >= 19);

            case "Canada.AB":
                return (p.getAge >= 18);
            ...
            }
        }
    }
```

Still some coupling
(AlcoholRegulation depends on Person)
More complex design :(

ALTERNATIVE 4

```
public class AlcoholRegulations {
public boolean canLegallyDrink(String country, int age){
    switch (country) {
        case "Afghanistan":
            return false;
        case "Armenia":
        return true;
        case "Canada.BC":
            return (age >= 19);
        case "Canada.AB":
        return (age >= 18);
    }
}
}
```

Demeter's principle respected

No more dependencies, rely on the use of basic data types.

POSSIBLE SOLUTION

Exercise 2 – What's wrong here ?

```
public class GameLauncher {
    protected RiskController controller;
    protected RiskGame game;
    private PrintWriter outChat = null;
    private BufferedReader inChat = null;
    private ChatArea chatter = null;
    private Socket chatSocket = null;
    private ChatDisplayThread myReader = null;
    private int applicationPort;
    protected String myIPAddress;
    protected boolean unlimitedLocalMode;
    private boolean autoplaceallMode;
    private boolean battleMode;
    private boolean replayMode;
    ...
}
```

TODO

Design a UML class diagram of this system where you would aim for more cohesion

Exercise 2 – What's wrong here ?

```
public class GameLauncher {
    protected RiskController controller;
    protected RiskGame game;
    private PrintWriter outChat = null;
    private BufferedReader inChat = null;
    private ChatArea chatter = null;
    private Socket chatSocket = null;
    private ChatDisplayThread myReader = null;
    private int applicationPort;
    protected String myIPAddress;
    protected boolean unlimitedLocalMode;
    private boolean autoplaceallMode;
    private boolean battleMode;
    private boolean replayMode;
    ...
}
```

CHAT USER INTEFACE

CHAT USER INTEFACE

CHAT USER INTEFACE

CHAT NETWORK

CHAT USER INTEFACE

NETWORK

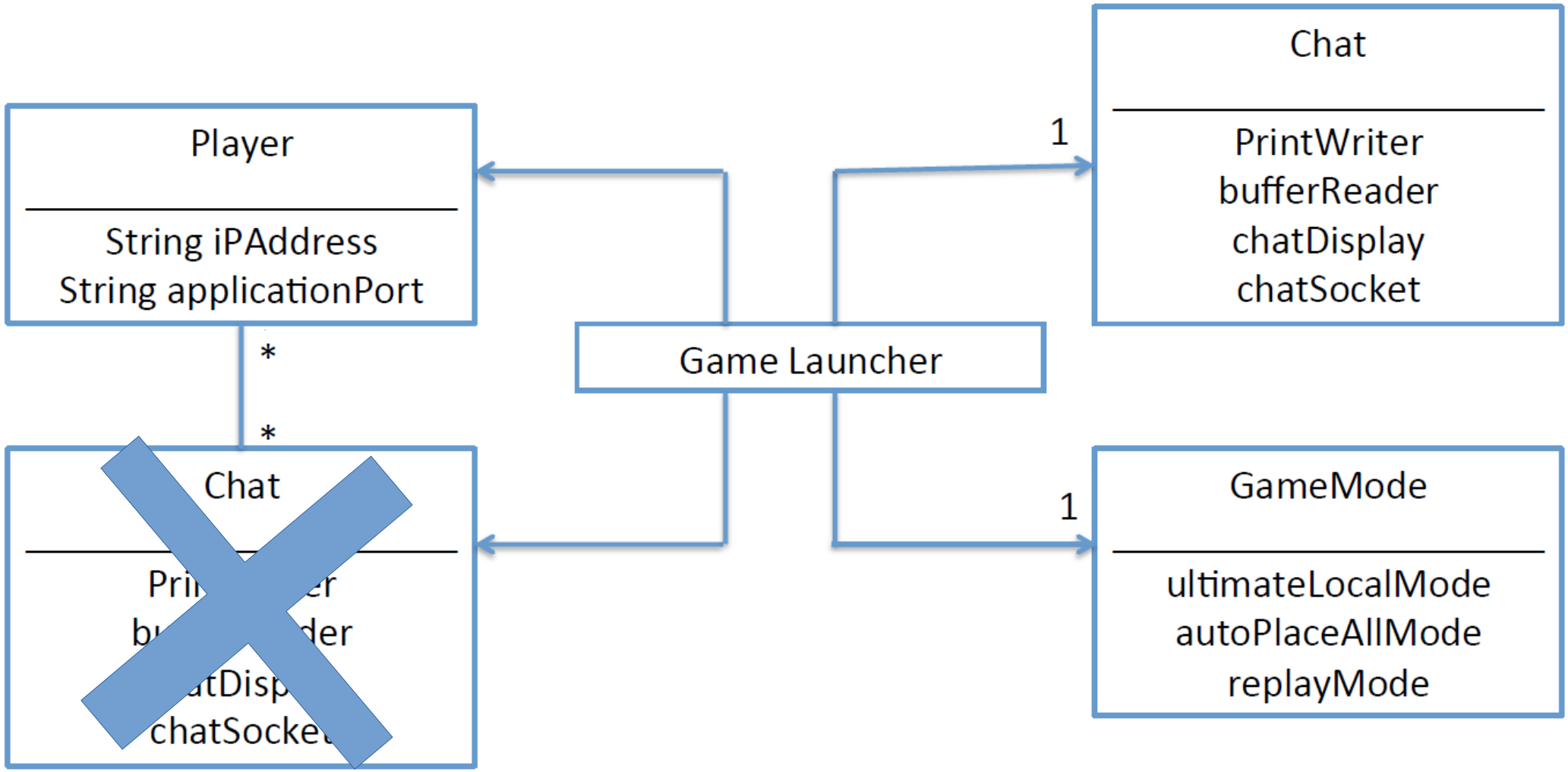
NETWORK

GAME PROPERTIES

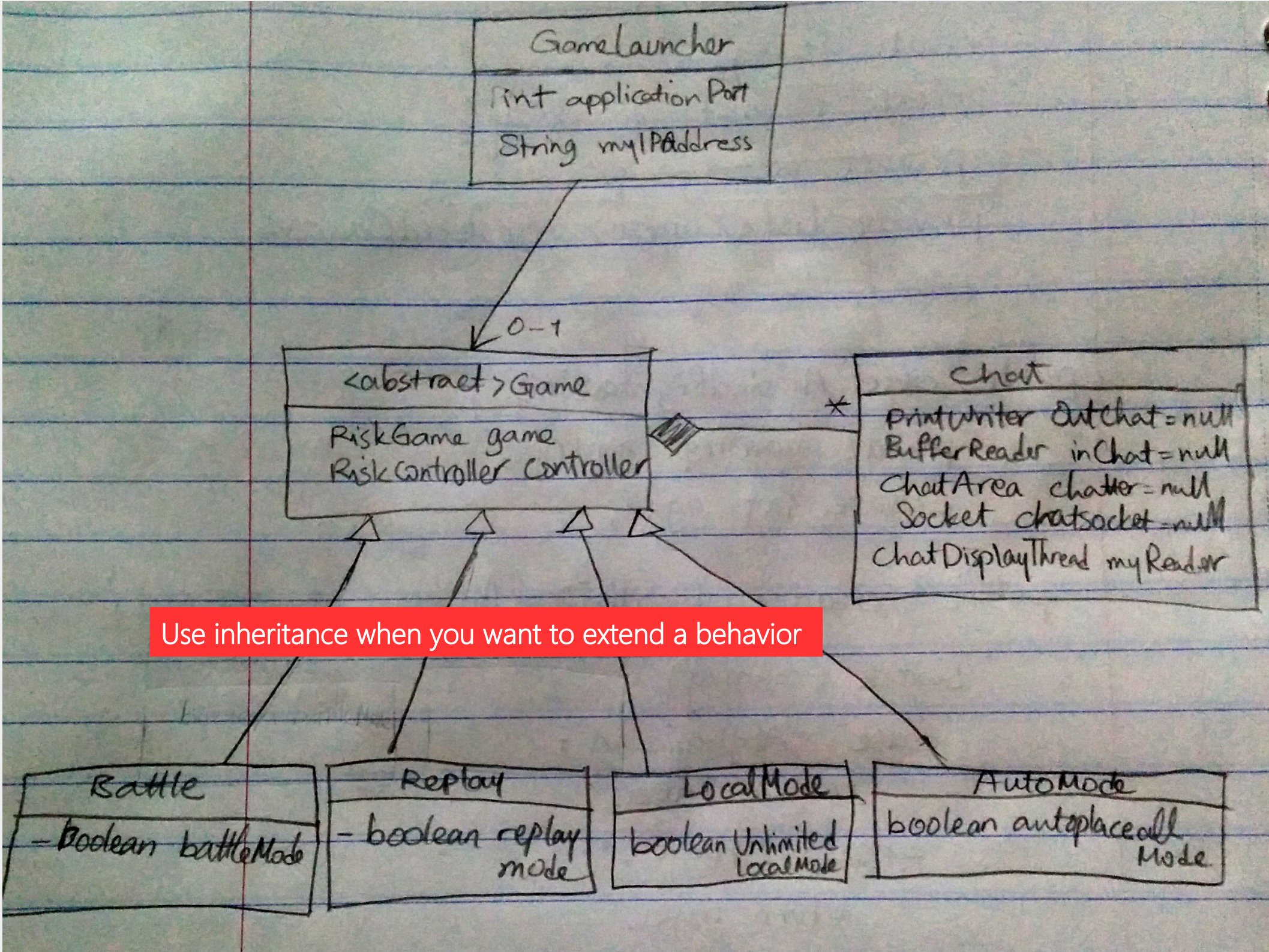
GAME PROPERTIES

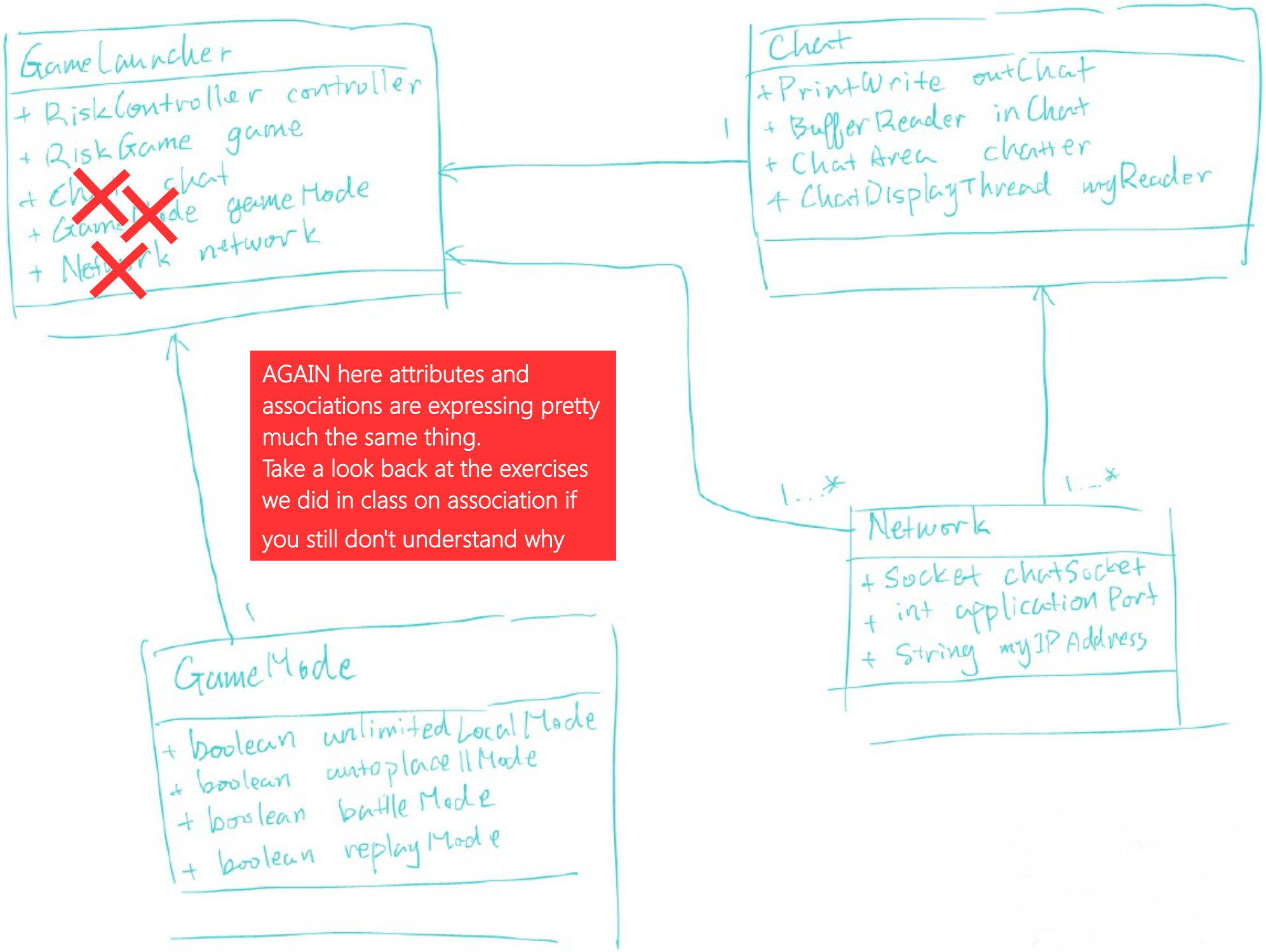
GAME PROPERTIES

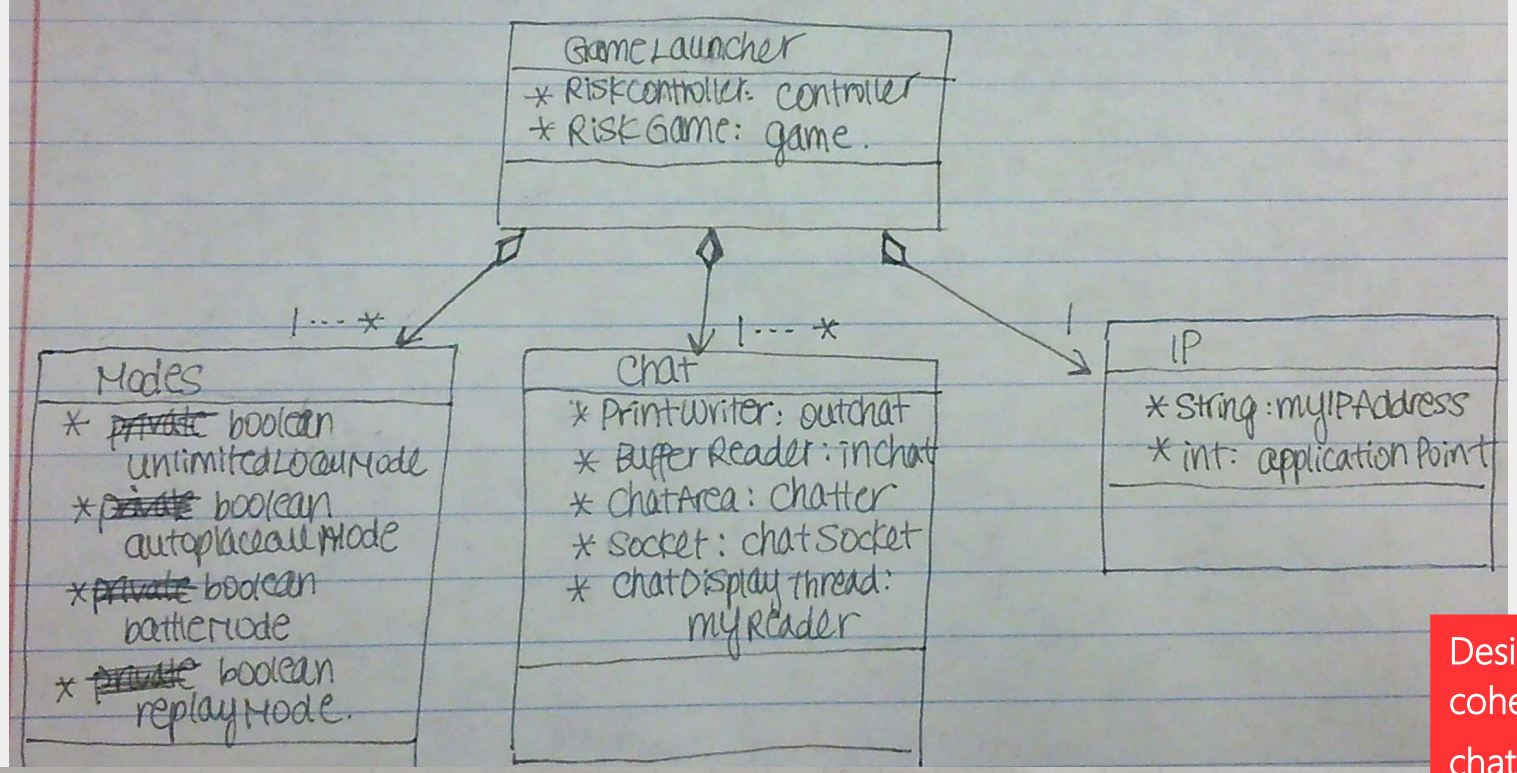
GAME PROPERTIES



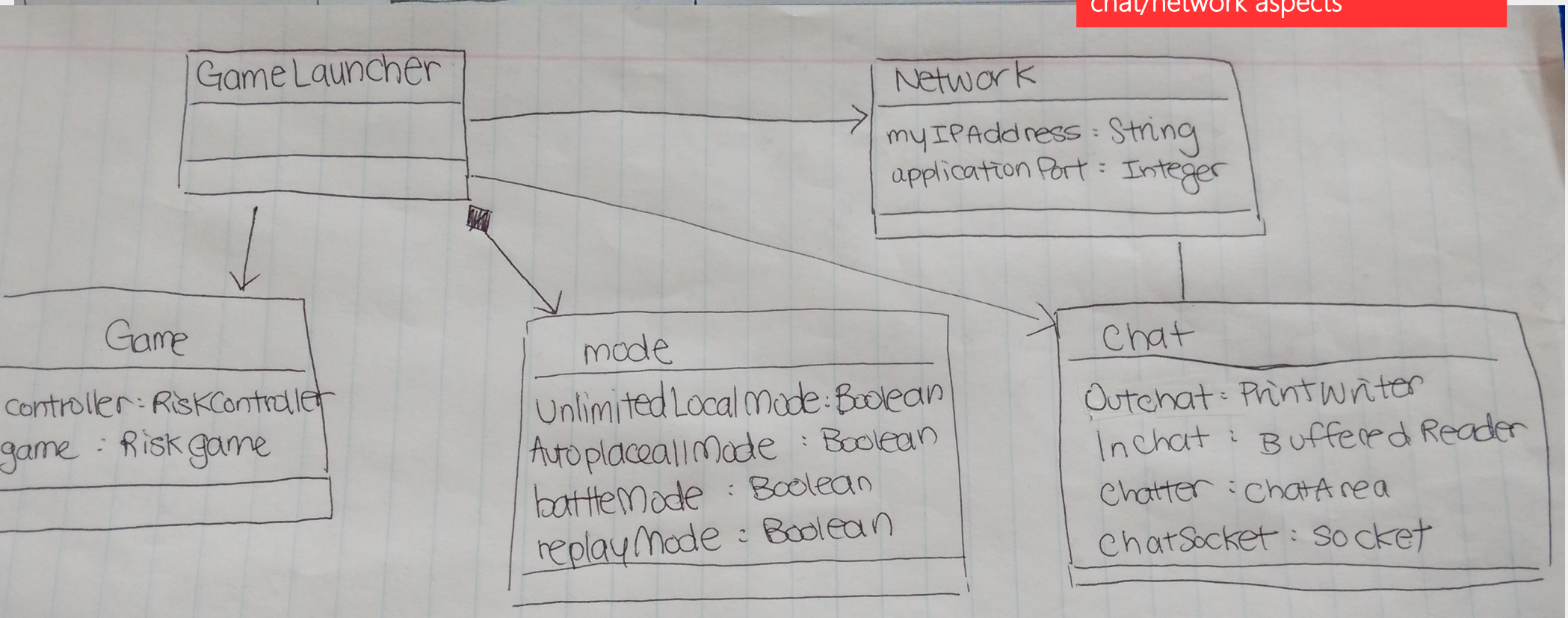
Do not duplicate classes, you can create several associations between classes if needed

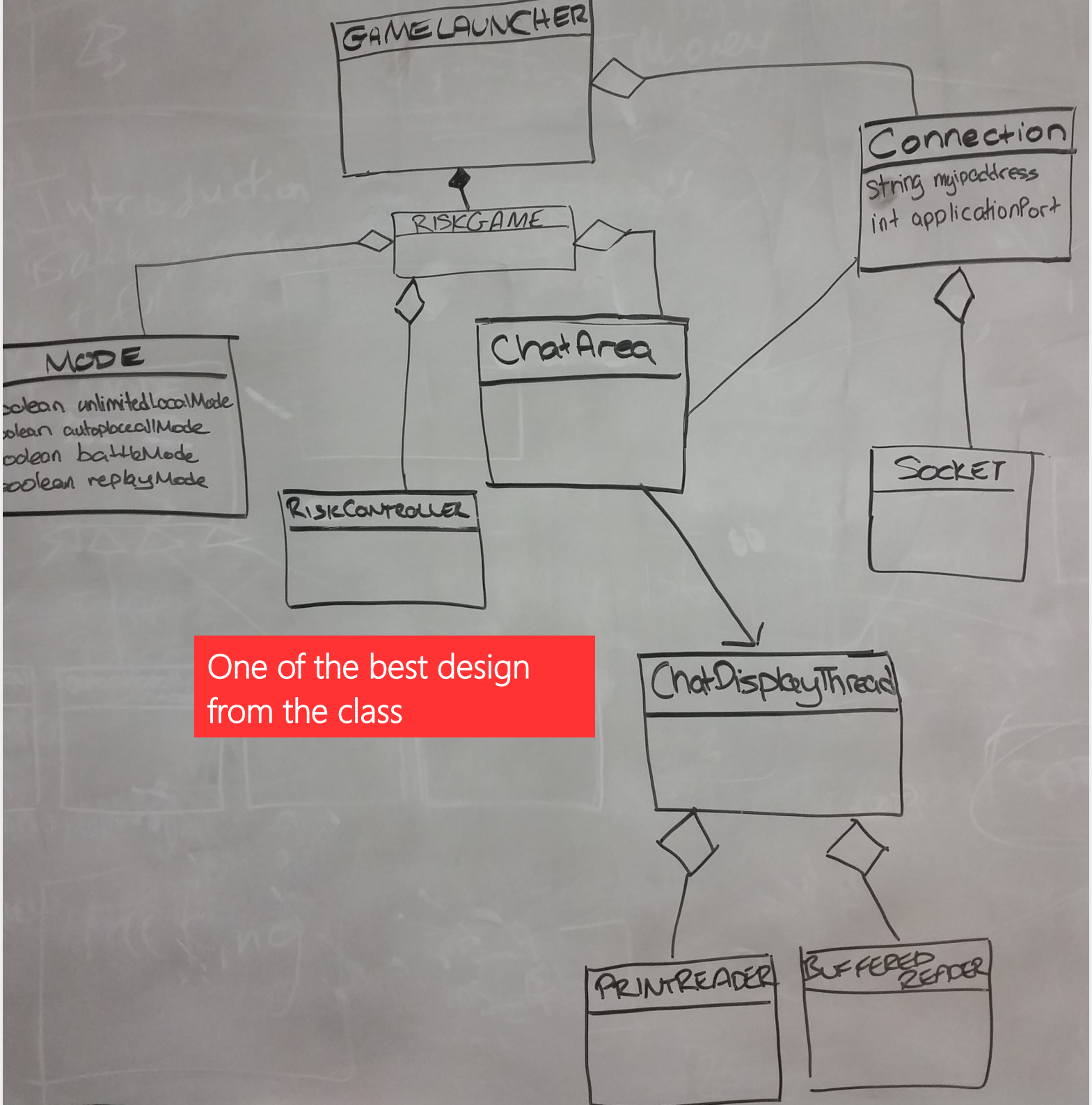

GameLauncher
int applicationPort
String myIPAddress
0-1
<abstract>Game
RiskGame game
RiskController controller
*
Chat
PrintWriter outChat=null
BufferReader inChat=null
ChatArea chatter=null
Socket chatSocket=null
ChatDisplayThread myReader
Use inheritance when you want to extend a behavior
Battle
-boolean battleMode
Replay
-boolean replay mode
LocalMode
boolean UnlimitedLocalMode
AutoMode
boolean autoplaceallMode


GameLauncher
+ RiskController controller
+ RiskGame game
+ Chat chat
+ GameMode gameMode
+ Network network
Chat
+ PrintWrite outChat
+ BufferReader inChat
+ ChatArea chatter
+ ChatDisplayThread myReader
1
AGAIN here attributes and associations are expressing pretty much the same thing.
Take a look back at the exercises we did in class on association if you still don't understand why
1..*
1..*
Network
+ Socket chatSocket
+ int applicationPort
+ String myIPAddress
1
GameMode
+ boolean unlimitedLocalMode
+ boolean autoplaceIIMode
+ boolean battleMode
+ boolean replayMode



Design pretty okay, still some cohesion problems with chat/network aspects




GAMELAUNCHER
Connection
String myipaddress
int applicationPort
RISKGAME
MODE
oolean unlimitedLocalMode
olean autoplaceallMode
oolean battleMode
oolean replayMode
ChatArea
SOCKET
RISKCONTROLLER
One of the best design from the class
ChatDisplayThread
PRINTREADER
BUFFERED READER