

# Avoiding Excess Factory Abstractness

---

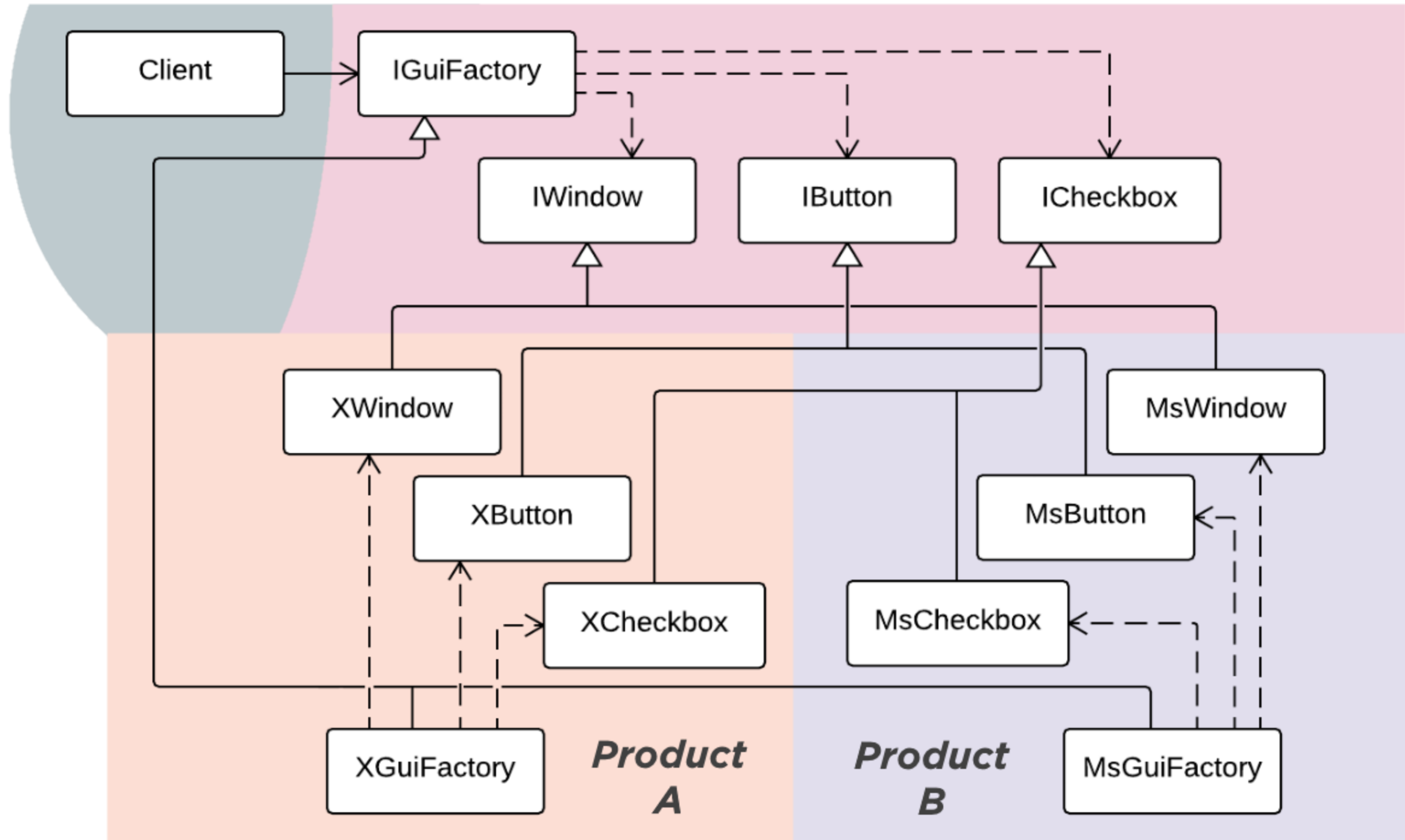


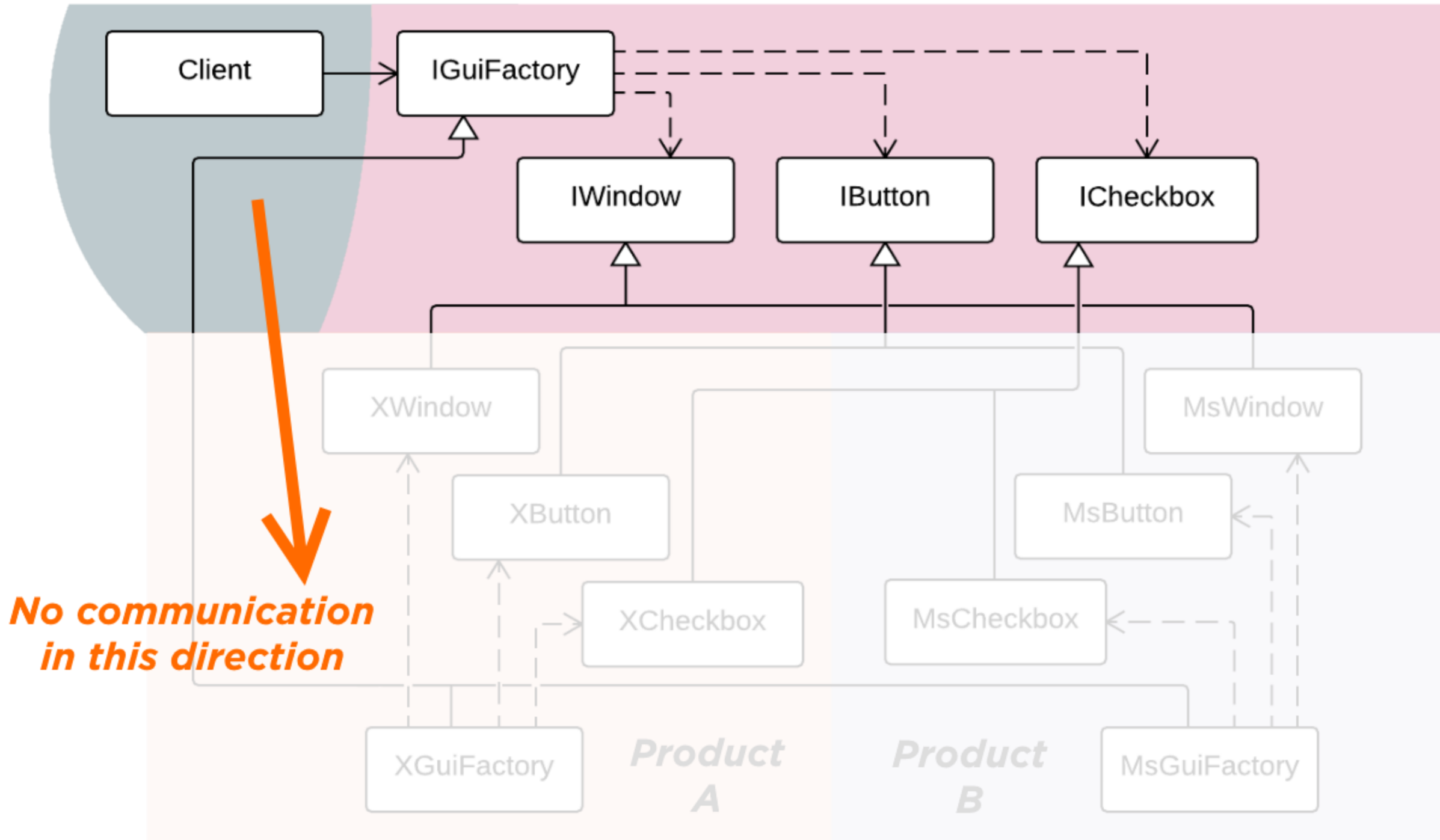
**Zoran Horvat**

OWNER AT CODING HELMET CONSULTANCY

@zoranh75 [www.codinghelmet.com](http://www.codinghelmet.com)







# Complications Inherent to Abstract Factory

**Abstract layer  
only contains  
features  
common to all  
concrete products**

**Some  
concrete products  
are yet to be  
discovered  
in the future**

**Arguments to  
Create method  
must satisfy all  
concrete products**



# Concrete Products May Depend on Each Other

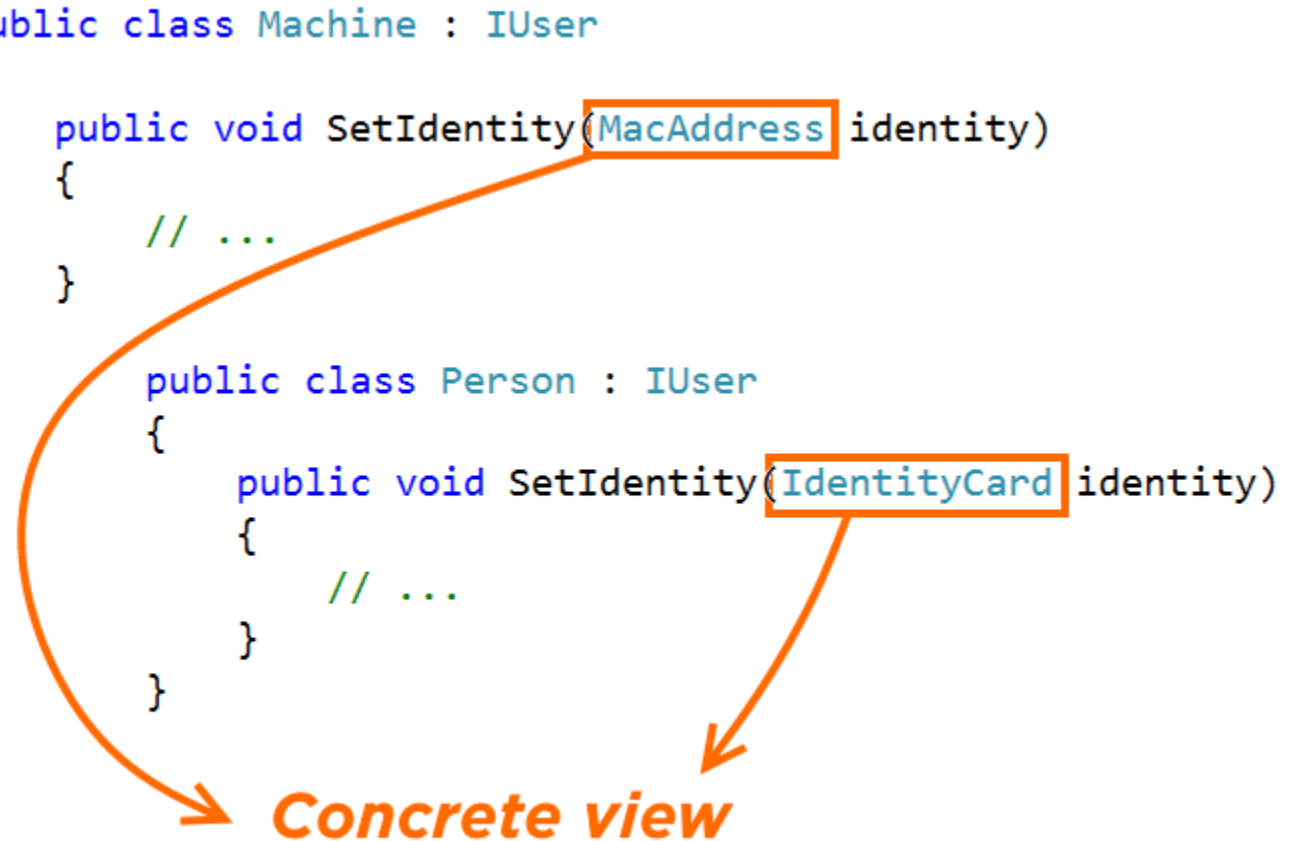
```
public interface IUser
{
    void SetIdentity(IUserIdentity identity);
}
```

***Abstract view***

```
public class Machine : IUser
{
    public void SetIdentity(MacAddress identity)
    {
        // ...
    }
}

public class Person : IUser
{
    public void SetIdentity(IdentityCard identity)
    {
        // ...
    }
}
```

***Concrete view***



# Abstract Factory Trade-offs

## **Conformist approach**

Make method arguments more abstract

Accept strings instead of concrete types

Stringly-typed solution may fail at runtime!

## **Escapist approach**

Support only one family of concrete products

Don't try to find what different families have in common

Move some concrete features into abstract part of the diagram



# Practical Problems with Abstract Factory

**Fixing one complication  
makes another grow bigger**

**Conclusion is to seek for the  
middle solution  
in each practical case**



# Trade-off

1. **Introduction**  
 2. **Background**  
 3. **Methodology**  
 4. **Results**  
 5. **Conclusion**  
 6. **References**  
 7. **Appendix**  
 8. **Index**  
 9. **Table of Contents**  
 10. **Summary**  
 11. **Abstract**  
 12. **Keywords**  
 13. **Subject Headings**  
 14. **Notes**  
 15. **Footnotes**  
 16. **References**  
 17. **Appendix**  
 18. **Index**  
 19. **Table of Contents**  
 20. **Summary**  
 21. **Abstract**  
 22. **Keywords**  
 23. **Subject Headings**  
 24. **Notes**  
 25. **Footnotes**  
 26. **References**  
 27. **Appendix**  
 28. **Index**  
 29. **Table of Contents**  
 30. **Summary**  
 31. **Abstract**  
 32. **Keywords**  
 33. **Subject Headings**  
 34. **Notes**  
 35. **Footnotes**  
 36. **References**  
 37. **Appendix**  
 38. **Index**  
 39. **Table of Contents**  
 40. **Summary**  
 41. **Abstract**  
 42. **Keywords**  
 43. **Subject Headings**  
 44. **Notes**  
 45. **Footnotes**  
 46. **References**  
 47. **Appendix**  
 48. **Index**  
 49. **Table of Contents**  
 50. **Summary**  
 51. **Abstract**  
 52. **Keywords**  
 53. **Subject Headings**  
 54. **Notes**  
 55. **Footnotes**  
 56. **References**  
 57. **Appendix**  
 58. **Index**  
 59. **Table of Contents**  
 60. **Summary**  
 61. **Abstract**  
 62. **Keywords**  
 63. **Subject Headings**  
 64. **Notes**  
 65. **Footnotes**  
 66. **References**  
 67. **Appendix**  
 68. **Index**  
 69. **Table of Contents**  
 70. **Summary**  
 71. **Abstract**  
 72. **Keywords**  
 73. **Subject Headings**  
 74. **Notes**  
 75. **Footnotes**  
 76. **References**  
 77. **Appendix**  
 78. **Index**  
 79. **Table of Contents**  
 80. **Summary**  
 81. **Abstract**  
 82. **Keywords**  
 83. **Subject Headings**  
 84. **Notes**  
 85. **Footnotes**  
 86. **References**  
 87. **Appendix**  
 88. **Index**  
 89. **Table of Contents**  
 90. **Summary**  
 91. **Abstract**  
 92. **Keywords**  
 93. **Subject Headings**  
 94. **Notes**  
 95. **Footnotes**  
 96. **References**  
 97. **Appendix**  
 98. **Index**  
 99. **Table of Contents**  
 100. **Summary**  
 101. **Abstract**  
 102. **Keywords**  
 103. **Subject Headings**  
 104. **Notes**  
 105. **Footnotes**  
 106. **References**  
 107. **Appendix**  
 108. **Index**  
 109. **Table of Contents**  
 110. **Summary**  
 111. **Abstract**  
 112. **Keywords**  
 113. **Subject Headings**  
 114. **Notes**  
 115. **Footnotes**  
 116. **References**  
 117. **Appendix**  
 118. **Index**  
 119. **Table of Contents**  
 120. **Summary**  
 121. **Abstract**  
 122. **Keywords**  
 123. **Subject Headings**  
 124. **Notes**  
 125. **Footnotes**  
 126. **References**  
 127. **Appendix**  
 128. **Index**  
 129. **Table of Contents**  
 130. **Summary**  
 131. **Abstract**  
 132. **Keywords**  
 133. **Subject Headings**  
 134. **Notes**  
 135. **Footnotes**  
 136. **References**  
 137. **Appendix**  
 138. **Index**  
 139. **Table of Contents**  
 140. **Summary**  
 141. **Abstract**  
 142. **Keywords**  
 143. **Subject Headings**  
 144. **Notes**  
 145. **Footnotes**  
 146. **References**  
 147. **Appendix**  
 148. **Index**  
 149. **Table of Contents**  
 150. **Summary**  
 151. **Abstract**  
 152. **Keywords**  
 153. **Subject Headings**  
 154. **Notes**  
 155. **Footnotes**  
 156. **References**  
 157. **Appendix**  
 158. **Index**  
 159. **Table of Contents**  
 160. **Summary**  
 161. **Abstract**  
 162. **Keywords**  
 163. **Subject Headings**  
 164. **Notes**  
 165. **Footnotes**  
 166. **References**  
 167. **Appendix**  
 168. **Index**  
 169. **Table of Contents**  
 170. **Summary**  
 171. **Abstract**  
 172. **Keywords**  
 173. **Subject Headings**  
 174. **Notes**  
 175. **Footnotes**  
 176. **References**  
 177. **Appendix**  
 178. **Index**  
 179. **Table of Contents**  
 180. **Summary**  
 181. **Abstract**  
 182. **Keywords**  
 183. **Subject Headings**  
 184. **Notes**  
 185. **Footnotes**  
 186. **References**  
 187. **Appendix**  
 188. **Index**  
 189. **Table of Contents**  
 190. **Summary**  
 191. **Abstract**  
 192. **Keywords**  
 193. **Subject Headings**  
 194. **Notes**  
 195. **Footnotes**  
 196. **References**  
 197. **Appendix**  
 198. **Index**  
 199. **Table of Contents**  
 200. **Summary**  
 201. **Abstract**  
 202. **Keywords**  
 203. **Subject Headings**  
 204. **Notes**  
 205. **Footnotes**  
 206. **References**  
 207. **Appendix**  
 208. **Index**  
 209. **Table of Contents**  
 210. **Summary**  
 211. **Abstract**  
 212. **Keywords**  
 213. **Subject Headings**  
 214. **Notes**  
 215. **Footnotes**  
 216. **References**  
 217. **Appendix**  
 218. **Index**  
 219. **Table of Contents**  
 220. **Summary**  
 221. **Abstract**  
 222. **Keywords**  
 223. **Subject Headings**  
 224. **Notes**  
 225. **Footnotes**  
 226. **References**  
 227. **Appendix**  
 228. **Index**  
 229. **Table of Contents**  
 230. **Summary**  
 231. **Abstract**  
 232. **Keywords**  
 233. **Subject Headings**  
 234. **Notes**  
 235. **Footnotes**  
 236. **References**  
 237. **Appendix**  
 238. **Index**  
 239. **Table of Contents**  
 240. **Summary**  
 241. **Abstract**  
 242. **Keywords**  
 243. **Subject Headings**  
 244. **Notes**  
 245. **Footnotes**  
 246. **References**  
 247. **Appendix**  
 248. **Index**  
 249. **Table of Contents**  
 250. **Summary**  
 251. **Abstract**  
 252. **Keywords**  
 253. **Subject Headings**  
 2

## strained naturally in some case?

## trade type safety for it!



```
public interface IControllerFactory
{
    IController CreateController(RequestContext requestContext, string controllerName);
    void ReleaseController(IController controller);
}
```

```
public interface IControllerFactory
{
    IController CreateController(RequestContext requestContext, string controllerName);
    void ReleaseController(IController controller);
}
```

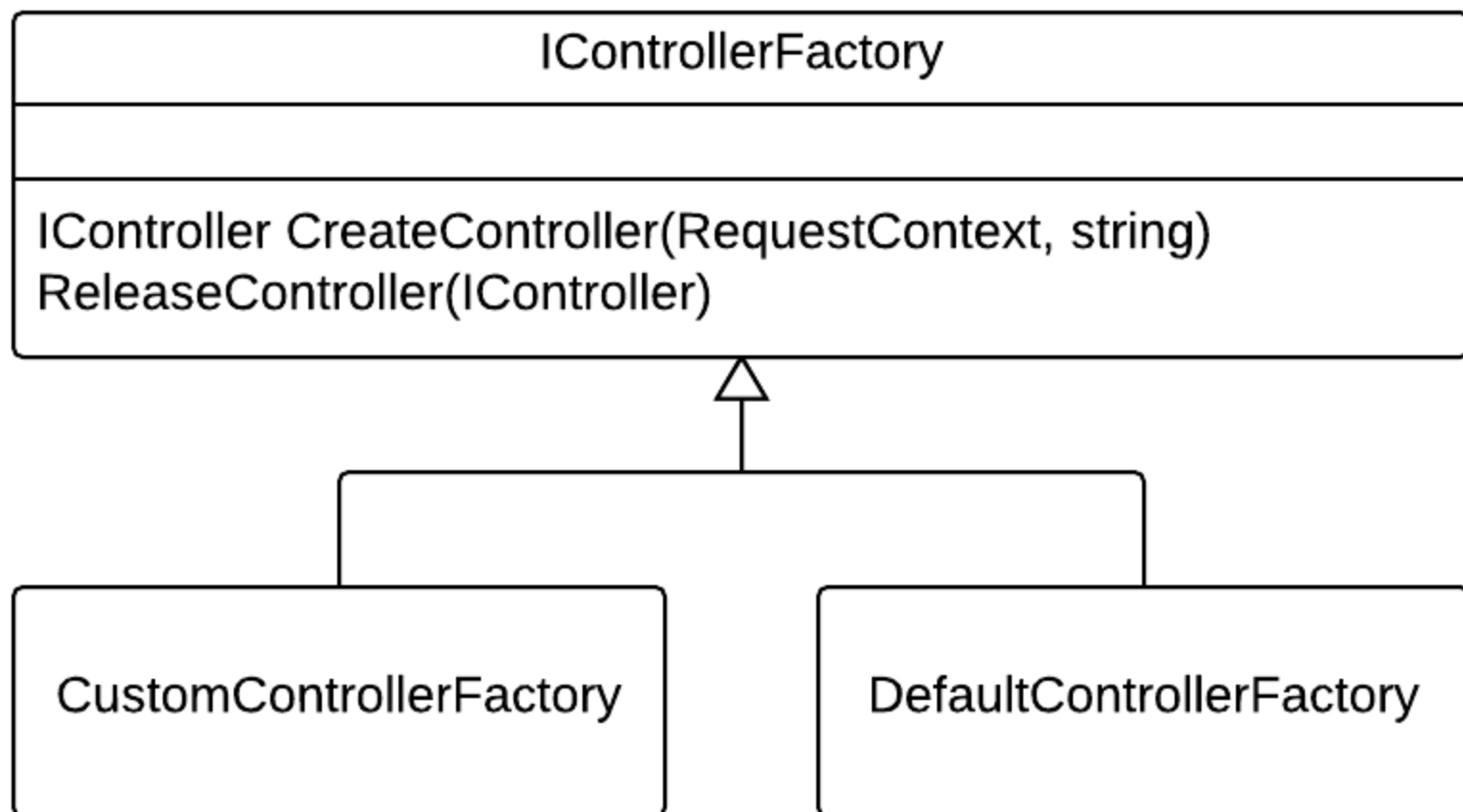
***Context is tightly bound  
to Web request***

***String used to support  
future classes***

***This abstract factory now supports  
only one family of concrete products***

***Multiplatform implementation is almost impossible  
even for this one family of products***

```
public interface IControllerFactory
{
    IController CreateController(RequestContext requestContext, string controllerName);
    void ReleaseController(IController controller);
}
```



# When we rely on strings...

**We cannot stop execution early when string is incorrect**

**String is passed to deeper layers for further analysis**

**The invalid string can even reach the Domain Model**

**Domain Model finally fails, but that is way too late**



# When we rely on strings...

**We loose  
compile-time  
checks**

**We loose  
run-time checks  
at early stages**

**We can just fail  
deep inside the  
application**



# HTTP Status Code in the Controller Factory?

## **Factory is doing it's caller's job**

Factory attempts to  
give clues about how  
to recover from  
an error

## **Factory is sending potentially confusing message**

It has analyzed the  
request

It has found the error

And it advises the  
caller how to recover

## **Error message sent from the Factory is too specific**

What if it shouldn't be  
status 404  
in this case?

What if it was the  
Internal Server Error,  
status 500 in the end?



```
public IController CreateController(RequestContext requestContext, string controllerName)
{
    if (controllerName != "Home")
        throw new HttpException((int)HttpStatusCode.NotFound, "Controller not found.");

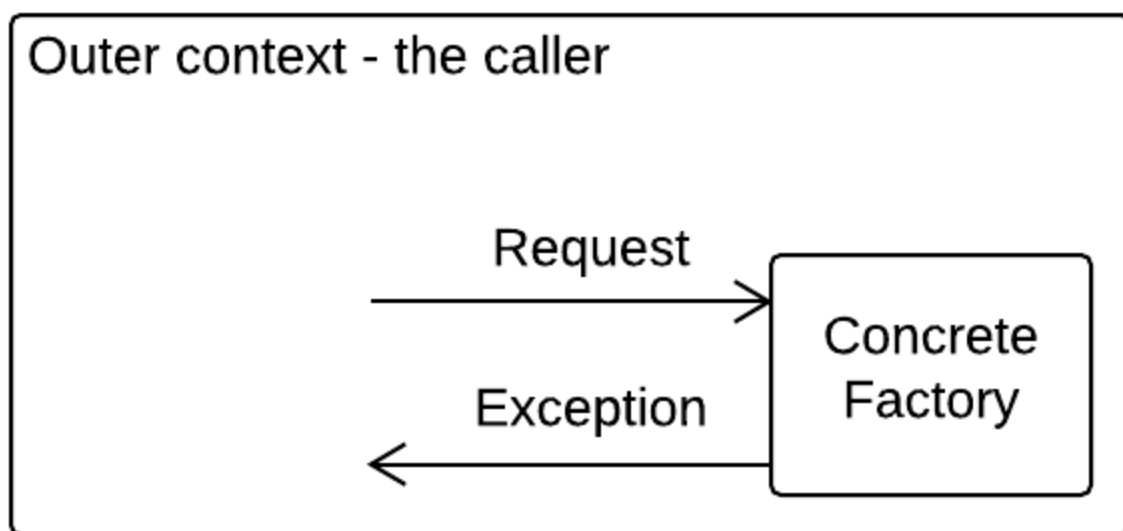
    return new HomeController(
        new AppSignature()
        {
            ApplicationName = "Concrete Factory Demo",
            AuthorName = "Zoran Horvat"
        });
}
```

```
public IController CreateController(RequestContext requestContext, string controllerName)
{
    if (controllerName != "Home")
        throw new HttpException((int)HttpStatusCode.NotFound, "Controller not found.");

    return new HomeController(
        new AppSignature()
        {
            ApplicationName = "Concrete Factory Demo",
            AuthorName = "Zoran Horvat"
        });
}
```

***HTTP status 404***

*Is this just a lucky guess?*





# Division of Responsibilities

Caller

Callee

```
if (controllerName != "Home")  
    throw new HttpException((int)HttpStatusCode.NotFound, "Controller not found.");
```

***Throw ArgumentException instead?***



# Consequences of Using Strings Instead of Concrete Objects

## **Stringly-typed coding**

Practice of passing a string  
when we don't have  
a more appropriate type

Abstract Factory often  
forces us to pass strings

There is no common type system  
for different product families

## **Potential solution**

Use Builder or Specification  
instead

These will hide concrete objects  
from the caller  
through encapsulation

No more strings will be required



# Summary



## Abstract Factory applied to a single family of concrete products

- This is opposed to trying to unite different families of products

## Positive consequences

- Some concrete features can be moved up the hierarchy into abstract types
- No fear that some derived product will suffer
  - There are no other derived products!

# Summary



## Other consequences

- There will be only one constructor
- No need to look for common constructor arguments
- The design is simpler and easier to use

# Summary



## An example

- Controller factory in ASP.NET MVC
- The framework supports only one family of controllers
- Create method arguments were meaningful to this family of products

## Stringly-typed code

- Controller was identified by a string
- This has caused instability



# Summary



## Error recovery from a concrete factory

- It was not a good idea to recover from feature supplier
- Recovery may require more information
- Called object has no such information
- The only one who may have it is the caller



## Summary



### Then how to use Abstract Factory well?

- By understanding the root cause of the issues first!

#### Root causes:

- Class dependencies
- Substitution principle
- Liskov substitution principle
- Method preconditions



#### ***Next module -***

*Understanding Dependencies,  
Covariance and Contravariance*

