Hidden Coupling

Example 1

Question

```
public void foo() {
  bar(1,2,3);
}

public void bar(int a, int b, int c) {
  System.out.println(a+b+c);
}
```

Answer

What is the coupling in this example? method call

Is the coupling visible or hidden? Visible coupling

If the coupling is hidden, then how can I remove it? n/a

Example 2

Question

```
char *str = "Hello, ";
char *buf = malloc(8+strlen(name));
strcat(buf, str, name);
```

Answer

What is the coupling in this example? Hidden coupling between 8 and the length of str;

Also hidden coupling between the strcat and the buffer length computation;

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Change 8 to `1+strlen(str)` .

Merge strcat and the buffer length computation with asprintf("%s%s", str, name)

Example 3

```
struct Game {
   ....
   Player players[2];
   ....
}


for (int i = 0; i < 2; i++) {
  if (game.players[i].isVictorious()) {
      ....
  }
}
```

Answer

What is the coupling in this example? Hidden coupling in the constant "2"

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Use a NUM_PLAYERS constant or (preferably) NELEM(game.players)

Example 4

```
def user_history(days=90):
  # Do 90 days of history by default
  for i in xrange(days):
    # do stuff
```

Answer

What is the coupling in this example? Hidden coupling with the number 90 between the comment and the code

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Eliminate the comment

Example 5

```
public class A {
  public void log() {
```

```
    ....
    writeLineToFile("log.txt", ...)
    ....
  }
}



public class B {
  public void b() {
    ....
    writeLineToFile("log.txt", ...)
    ....
  }
}
```

Answer

What is the coupling in this example? Hidden coupling in the filename

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Encapsulate behind a method or use a shared constant

Example 6

Client

```
<form ...>
  <input type="text" name="username"></input>
  <input type="text" name="password"></input>
  <button type="submit"></button>
</form>
```

Server

```
def handleRequest(request):
  username = request["username"]
  password = request["password"]
  # …
```

Answer

What is the coupling in this example? Hidden coupling in the form field names

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Use a templating engine to generate HTML instead.

Example 7

```
openFile("prog/imgs/combat_images/MONSTER1.gif");
openFile("prog/imgs/combat_images/FIREBALL.gif");
```

Answer

What is the coupling in this example? Hidden coupling in the string prefixes / folder paths

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Use functions to generate full filepaths.

Example 8

```
class Rectangle { public int getArea() { ... } }
class Circle { public int getArea() { ... } }

class GraphicsProgram {
   ...
   private Color computeAverageColor() {
     for (Rectangle r : this.rectangles) { ... }
     for (Circle c : this.circles) { ... }
   }
}
```

Answer

What is the coupling in this example? Hidden coupling in the interfaces of the Rectangle and Circle classes. A single design change may alter both of them: if you add getPerimeter() to Circle because of some decision, you are very likely to also make the same decision to do so for Rectangle.

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Put behind a common interface

This is the hardest example in this assignment; a small fraction of students get it.

In Jimmy's first project after he began thinking deeply about software engineering principles, he had two modules for managing different data repositories; these were clones of each other, as there was no a priori reason they should inherit the same functionality. But in fact, he found he kept making the same enhancements to each. Insights from that project eventually led to the formation of the idea of hidden coupling. It inspired this example right here, and inspired to the "makeStudentEmployee" example in "The Design of Software is a Thing Apart."

Example 9

```
String[] recipeNames     = { "Fried calamari", "Spaghetti with
 ↪  meatballs", "Apple pie", ... };
RecipeType[] recipeTypes = {APPETIZER, ENTREE, DESSERT, ... }
```

Answer

What is the coupling in this example? Hidden coupling in the array indices

Is the coupling visible or hidden? Hidden coupling

If the coupling is hidden, then how can I remove it? Merge them into an array of structures.

X Macro question

Answer

Similar to example 9; they are spreading information about the DEX instructions among several arrays, case statements, and similar artifacts, so they need to have them generated from a single source.

More detailed answer from a former student:

Applying the X-Macro trick to the first macro group, we have the following expansion:

```
const char* const Instruction::kInstructionNames[] = {
#define INSTRUCTION_NAME(o, c, pname, f, r, i, a, v) pname,
#include "dex_instruction_list.h"
  DEX_INSTRUCTION_LIST(INSTRUCTION_NAME)
#undef DEX_INSTRUCTION_LIST
#undef INSTRUCTION_NAME
};
expands to:

const char* const Instruction::kInstructionNames[] = {
  "nop",
  "move",
  "move/from16",
```

```
  ...
};
```

The X-Macro trick here is used to efficiently initialize parallel lists of constants. If const arrays were used directly in code, then the arrays would be coupled together (changing one element in any array would cause a change in every other const array). To avoid this problem, we logically group the constants together inside an X-Macro definition, and generate the constants in the precompilation step. We remove the hidden coupling and end up with no run-time performance penalty because const arrays are very space-efficient.