Final Exam

Developing Web Applications

Submitted by,

Basavraj Jaliminche

8800149

**HTML / CSS / JavaScript**

**Q.1.**

**a. Using your knowledge of *semantic* html, provide an improved version of the following html script. (4 points).**

|  |  |
| --- | --- |
| **Original Script** | **Your Improved Script** |
| <html>  <head>  <title>The Title</title>  </head>  <div class=”container”>  The main part of the file  </div>  <div class=”bottom”>  The bottom part of the file  </div> | <!DOCTYPE html>  <html>  <header>  <title>The Title</title>  </header>  <body>  <div class="containter">  <h2><b><i><center>The main part of the file</center></i></b></h2>  </div>    <div class="bottom">  <h3><center>The bottom part of the file</center></h3>  </div>  </body>  </html> |

**b. In your words, what is the main advantage of using semantic html instead of non-semantic html?**

* Semantic HTML refers to a syntax that makes HTML easier to understand by better defining different parts and layouts of web pages.
* It makes the website more informative and adaptable, allowing browsers and search engines to better interpret the content.

**c. (Fill in the blank: The above “original script” runs correctly in most browsers. However, the first line does not contain a <!DOCTYPE html>** **to allow html parsing applications (such as a web browsers) know that this is an html document.**

**Q.2 (3 points) A beginner JavaScript coder attempted to get the console to write “You got me!” upon running the html file below. However, she can only see `null` outputted to the screen.**

|  |
| --- |
| JavaScript file |
| <html>  <head>  <script>  var elem = document.getElementById(‘grabme’);  console.log(elem.textContent);  </script>  </head>  <div class=”container” id=”grabme”>  You got me!  </div>  </html> |

**a.(2 points) Explain in your words why `elem` is equal to `null` instead of `You got me!` in this program.**

Ans :

* This error seems to indicate that there is no such element with the ID passed to getElementById().
* This can happen when JavaScript code is executed before the page is fully loaded, so the element cannot be found.
* The solution is that you must insert the JavaScript code after the closing HTML element or more generally before the </body> tag.

**b.** **(3 points) Provide two different ways to correctly output whatever text is inside the #grabme identified element. You may explain in your own words or provide an example script.**

1. **First way. (1.5 points)**

The easiest way to modify the content of an HTML element is by using innerHTML property. To change the content of HTML element, we use the following syntax:

document.getElementById().innerHTML = new HTML

1. **Second Way way. (1.5 points)**

Second method with which we can we can output the text within the grab is the onclick function of JavaScript. The function can be used as:

<button type="button" onclick="document.write(‘grabme’)">Button Text</button>

**Q.3 (3 points) For the next questions on CSS selectors, refer to the following html file:**

|  |
| --- |
| <html>  <head>  </head>  <div id=”grandparent1”>  <div id=”parent1”>  <div id=”child1”>  <div id=”grandchild1” class=”g1”></div>  <div id=”grandchild2”></div>  </div>  <div id=”child2”>  <div id=”grandchild3” class=”g1”></div>  </div>  </div>  </div> |

**a. (1 point) Provide the CSS selector to choose all elements in the file.**

Ans: The selector \* selects all elements. \* selectors can also select all elements within another element.

**b.** **(1 point) Provide a CSS selector to include all elements that have an id that includes the substring “grand.”**

Ans: The id selector ( # ) selects HTML elements based on the value contained in their id attribute. Use an ID selector when you want to target a specific element on a web page.

**C.**

**Java Practice / Design Patterns Questions**

**Q.4 (6 points) For questions (a), (b) and (c), refer to the following Java classes:**

|  |  |
| --- | --- |
| interface IWidgetBuilder {  Widget build();  }  class WidgetBuilder extends IWidgetBuilder {  private Widget widget;  WidgetBuilder() {  this.setUp();  }  void setUp() {  this.widget = new Widget();  }  /\*\* Omitted code includes methods `addId`, `addName`, `addProbability`, `addDepth` and `addStyle`  to support the creation of a widget. \*\*/  }  @Entity  class Widget {  @Id  @GeneratedValue(strategy=GeneratorType.AUTO)  private int id;  private String name;  private double probability = 0.1;  private double depth;  private Style style;  Widget () {}  // getters and setters  }  class Style {  Style() {}  } |  |

1. **(3 points) Assuming that a WidgetBuilder is a properly formed Builder class that creates a Widget, provide a statement that calls the WidgetBuilder to produce a Widget object containing a name and style. (Assume depth will never be called). (3 points)**

**Ans :** WidgetBuilder (const Text(

"My Name",

style: TextStyle(fontStyle: FontStyle.italic),

))

1. **(2 points) Provide a sample `build` method for the WidgetBuilder class.**

Ans:

@override

Widget build(BuildContext context) {

return Container(

child: Row(

children: [

const Text(

"My Name",

style: TextStyle(fontStyle: FontStyle.italic), ),

], ), ); }

1. **(1 point) Fill in the blank. In general, we use builders to create objects to avoid building classes that have Constructors with too many parameters.**

**Q.5. (Total 15 points) The following questions (7a – 7e) will refer to the below poorly designed arrangement of classes. Assume that all classes are members of the same package (ie. There are no unidentified class errors due to class order).**

|  |
| --- |
| class PoorDesign1 {  private PoorDesign2 buoy;  private PoorDesign3 toy;  PoorDesign1 (String type, int temperature, double waveHeight,  String name, String description) {  this.toy = this.makeAToy(name, description);  this.buoy = toy.makeABuoy(type);  this.buoy.setTemperature(temperature);  this.buoy.setWaveHeight(waveHeight);    }  PoorDesign3 makeAToy(String name, String description) {  return new PoorDesign3(name, description, this.buoy.type);  }  String hello() {  return “hello”;  }  String goodbye() {  return “goodbye”;  }  }  class PoorDesign2 {  protected String type;  private int temperature = 10;  private double waveHeight = 10.0;  PoorDesign2(String type) {}  // standard getters and setters  }  class PoorDesign3 {  protected String type;  private String name;  private String description;  PoorDesign3(String name, String description, String type) {  this.name = name;  this.type = type;  this.description = description;  }  PoorDesign2 makeABuoy() {  return new PoorDesign2(this.type);  }  } |

1. **(2 points) Bhavani is one of your programming colleagues and a (fictional) former Prog 8610 *Web Application Development* student. She argues strongly that the methods `hello` and `goodbye` should be moved to another class. Do you agree? Explain why.**

**Ans:**

According to the systematic working of the Factory Method design approach, the parent class is responsible to provide interface for the creation of an object and subclasses decide which class to instantiate. Therefore, Bhavani is rightly arguing to move `hello` and `goodbye` from parent class to any other subclass.

**b.** Bhavani also thinks that PoorDesign1 should be called `ThingFactory` and that the constructor should accept no parameters when called. Similarly, PoorDesign2 should be called `Buoy` and PoorDesign3 should be called `Toy`. Bhavani also says `ThingFactory` should have a method called `makeThing(String type)`. If type is “buoy” it will return a `Buoy` class and if the type is “toy”, then it will return a `Toy` class. Otherwise the `makeThing` method returns null. To do this correctly, Bhavani says your team must also create a **makeThing method** that the Toy and Buoy classes will **call** if we want the Factory Method design approach to work properly.

**C .** **(2 points) There is a major bug in the above design such that when main() function contains `PoorDesign1 design = new PoorDesign1();` the package stops and returns an exception. Explain the exception and why it is occurring in this program.**

**Ans :**

*error: constructor PoorDesign1 in class PoorDesign1 cannot be applied to given types;*

*PoorDesign1 design = new PoorDesign1();*

*^*

*required: String,int,double,String,String*

*found: no arguments*

*reason: actual and formal argument lists differ in length*

As mentioned in above error, constructor definition is not matched. As in the class, a parameterized constructor is being used whereas in the main method, no arguments are passed which causes the above exception and halt the program.

1. (**2 points) There is a major bug in the above design such that when main() function contains `PoorDesign1 design = new PoorDesign1();` the package stops and returns an exception. Explain the exception and why it is occurring in this program.**

**Ans:** Test Driven Development (TDD) is software development approach in which test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free. Thus, TDD can be helpful to remove error found in part C, due to its unit testing approach.TDD helps to avoid any bug in a program in a number of ways. It fosters the creation of optimized code; it helps developers better analyze and understand client requirements and request clarity when they are not adequately defined. Moreover, the addition and testing of new functionalities become much easier in the latter stages of development. Test coverage under TDD is much higher compared to the conventional development models. This is because the TDD focuses on creating tests for each functionality right from the beginning. In addition to this, it also enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.

1. **The above design suffers from high coupling. Explain why this is the case. Use two examples that show how the design will create problems in the future when you want to add or change your code. (6 points)**

**Ans :**

Coupling is the measure of how dependent your code modules are on each other. Strong coupling is bad and low coupling is good. Heavily coupled code is difficult to reuse because it is difficult to remove from the system for use elsewhere. One should strive to reduce coupling in one’s code to as high a degree as possible. However, the Factory pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code. The use of factories in the code facilitates delegation of specific attributes of an object into specific subclasses. Therefore, the Factory pattern helps make a system independent of how objects are created. The above given code is highly coupled few examples are as follows:

***this.toy = this.makeAToy(name, description);***

***PoorDesign3 makeAToy(String name, String description) {***

***return new PoorDesign3(name, description, this.buoy.type);***

***}***

The above code of PoorDesign1 class is highly dependent upon PoorDesign3 class. In case of any modification in PoorDesign3 Class will compel PoorDesign1 class to make changes accordingly.

***PoorDesign2 makeABuoy() {***

***return new PoorDesign2(this.type);***

***}***

The above code of PoorDesign3 class is highly dependent upon PoorDesign2 class. In case of any modification in PoorDesign2 Class will compel PoorDesign3 class to make changes accordingly.

1. **Q.6** Create a form for students to sign up for one of three workshops scheduled at April 22, at 12pm. The form should include the students name, Student number, email, and payment method. When the form is submitted, an output box appears on the right side of the screen with preliminary information about the program.
   1. Once order if pressed, the payment is submitted and cannot be re-ordered.
   2. However, the user can retrieve information about their order using their studentID with the original form.