COMP 7012

# Exam 1

Spring 2019

Name: Solutions, ______________________
Last name — First name

**Rules:**

- No potty breaks.
- Turn off cell phones/devices.
- Closed book, closed note, closed neighbor.

- <u>WEIRD!</u> Do not write on the backs of pages. If you need more pages, ask me for some.

**Reminders:**

- Verify that you have all pages.
- Don't forget to write your name.
- Read each question <u>carefully</u>.
- Don't forget to answer <u>every</u> question.

1. [1%] Which of the following best describes Git?

   a. Distributed version control system
   b. Online version-control repository hosting service
   c. Virtual machine monitor
   d. Web development framework
   e. None of the above

2. [1%] Which of the following best describes GitHub?

   a. Distributed version control system
   b. Online version-control repository hosting service
   c. Virtual machine monitor
   d. Web development framework
   e. None of the above

3. [1%] Which of the following knowledge areas is Git most closely associated with?

   a. Software Design
   b. Software Construction
   c. Software Maintenance
   d. Software Configuration Management
   e. Software Engineering Process

Each of the following problems presents a Git log graph (log messages omitted) of a local repo and a scenario. Update the graph by crossing out and/or adding appropriate text.

- You may or may not need to use the two blank lines above the graph.
- If you need to add a commit, use the hash `c1c1c1c`.
- If a command would be rejected by GitHub (e.g., because the remote contains work that you do not have locally), write "REJECTED" on the top line.
- Assume that all remote bookmarks depicted are up to date.

I have included an example problem and solution below to help clarify what's expected.

**Example Problem**

Scenario: Developer makes changes to the code, stages the changes, and commits.

```


* 86b8116 (HEAD -> master, iss1)
* dc003f8
* 026c6cf
```

**Example Solution**

```

* c1c1c1c (HEAD -> master)
* 86b8116 (~~HEAD -> master,~~ iss1)
* dc003f8
* 026c6cf
```

4. [3%] Scenario: Developer runs `git checkout -b iss2`.

```
* 97c9227 (~~HEAD ->~~ master~~)~~, HEAD -> iss2)
* ed11409
* 137d7d0
```

5. [3%] Scenario: Developer makes changes to the code, stages the changes, and commits.

```
*   c1c1c1c (HEAD -> iss2)
|
| * 04ca8c6 (master)
* | 3283dd7 ~~(HEAD -> iss2)~~
|/
* a8da338
* fe2251a
```

6. [3%] Scenario: Developer runs `git checkout iss4`.

```
  * d197593 ~~(iss4)~~ (HEAD -> iss4)
* | 0f50aa4 (iss3)
|/
* 75a7005 (~~HEAD ->~~ master)
* cbff2e7
```

7. [3%] Scenario: Developer runs `git merge iss5`. Assume that auto-merge, if used, would complete successfully with no merge conflicts.

```
  * d3994b3 (iss6)
* | 0152ac4 (iss5, HEAD -> master)
|/
* 77a9025 (HEAD -> master)
* cdf1207
```

8. [1%] Would this be a fast-forward merge?

   a. Yes (circled)

   b. No

9. [3%] Scenario: Developer runs `git merge iss7`. Assume that auto-merge, if used, would complete successfully with no merge conflicts.

```
* c1c1c1c (HEAD -> master)
|\
| * g4ca8c6 (iss7)
* | 3283dd7 (HEAD -> master)
|/
* a8da338
* fe2251a
```

10. [1%] Would this be a fast-forward merge?

   a. Yes

   b. No (circled)

11. [3%] Scenario: Developer runs `git pull origin master`. Assume that auto-merge, if used, would complete successfully with no merge conflicts.

```
* a3b25e7 (origin/master, master, HEAD -> iss8)
* e41c5b6
* 9d63832 (~~HEAD -> iss8,~~ origin/iss8)
* 40f26d8
```

12. [3%] Scenario: Developer runs `git push`. Assume that auto-merge, if used, would complete successfully with no merge conflicts. Assume that all issue branches are tracking with their corresponding branches on the remote.

```
  * d3994b3 (HEAD -> iss9, origin/iss9)
* | 0152ac4 (origin/master, master)
|/
* 77a9025 ~~(origin/iss9)~~
```

13. [3%] Scenario: Developer runs `git push origin master`. Assume that auto-merge, if used, would complete successfully with no merge conflicts. Assume that all issue branches are tracking with their corresponding branches on the remote.

REJECTED

```
  * e28a3c2 (HEAD -> iss10, origin/iss10)
  * f40dd3b
* | 1061bb5 (origin/master, master)
|/
* 86b8116
```

Consider the following list of `git` commands:

- `add`
- `branch`
- `checkout`
- `clone`
- `commit`
- `init`
- `log`
- `merge`
- `pull`
- `push`
- `remote`
- `status`

Ginger has just joined a team of developers collaboratively working on a dog-walk sharing service called *Wufer*. The code for the project is housed in a GitHub repo. All work for the project is being done on issue branches and merged into a `master` branch.

14. [1%] Ginger wants to get a local copy of the code and repo, so she can begin contributing to the project. Which command from the above list should she use?

    clone

15. [1%] Ginger wants to create a new issue branch, `iss15`, and switch to the branch. Which command(s) from the above list should she use?

    checkout OR branch, checkout

16. [1%] Ginger makes some edits to the source code. Which command from the above list should she use to stage her changes?

    add

17. [1%] Now that she has staged her changes, which command from the above list should she run to save those changes to the local repo?

    commit

18. [1%] Having saved to her local repo, she would now like to upload her issue branch to GitHub. Which command from the above list should she run?

    push

19. [1%] While Ginger was working on her changes, other developers on the team merged new changes into the `master` branch. Which command from the above list should she run to merge those changes into her issue branch.

    pull

When she runs the command, she gets this message:

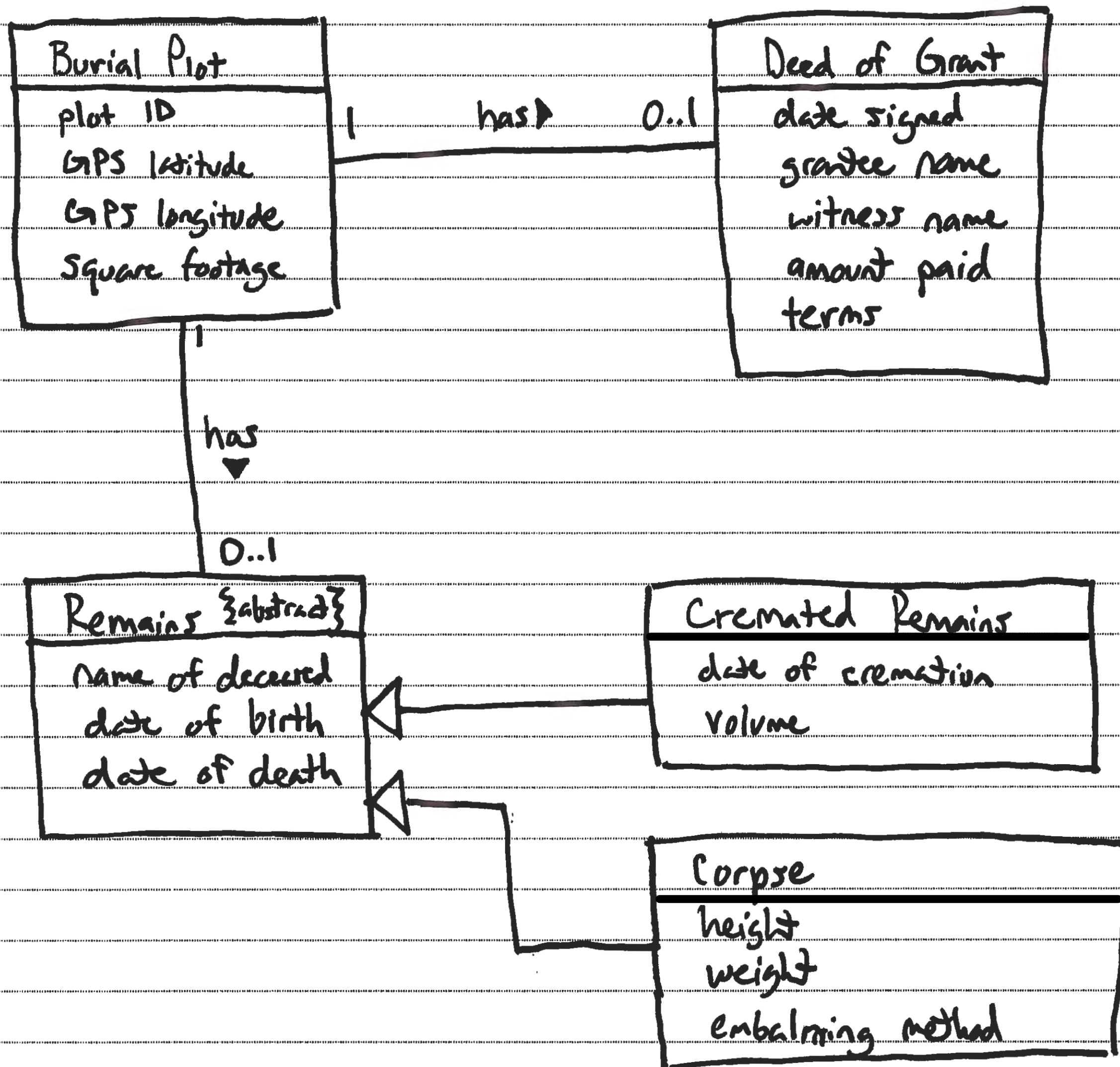
```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then ██████ the result.
```

20. [1%] Ginger edits the README file to resolve the conflict. Which command(s) from the above list should she run next to save these changes to her local and remote issue branches?

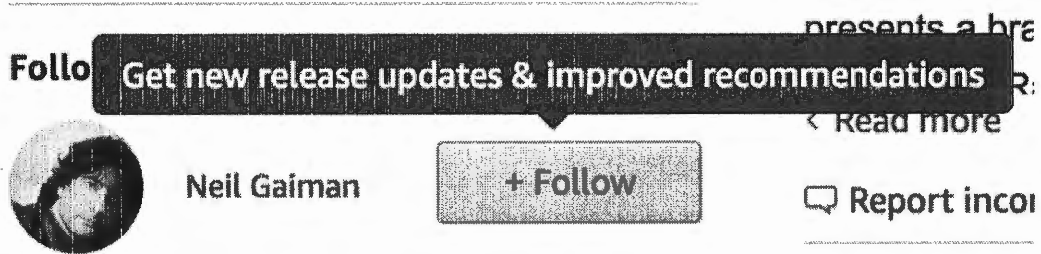
    add, commit, push

21. [15%] You have been tasked with creating a cemetery management system. Create a domain model (using UML class diagram notation) based on the following description of cemetery management. Model only things that are specifically described. Include all conceptual classes, attributes, associations, and generalization relationships mentioned. Label all associations, include all multiplicities, and denote any abstract classes. Do not explicitly model the "system".

There are a number of burial plot records. Each such record has the ID of the plot, the GPS latitude and longitude, and square footage of the plot. Some burial plots have a deed of grant. A deed of grant records a date signed, the name of a grantee, the name of a witness, the amount paid for the plot, and the terms of the grant. A burial plot may also have a set of remains. There are two types of remains: cremated remains and corpses. All remains, regardless of type, have the name of the deceased person, the person's date of birth, and their date of death. Cremated remains additionally have a date of cremation and a measure of volume. Corpses have a height, weight, and a description of embalming method.



When you view a book in Amazon, you will see this feature for the author of the book:




22. [6%] Reverse engineer one user story (title + description) that records a requirement for the above functionality. You must apply the templates described in class, and your US must have the other attributes of good user stories, which we discussed in class.

Title: Follow Author

Description: As a customer, I want to follow an author, so that I get new release updates and improved recommendations.

When you view an item in Amazon, you will see this feature:

In Stock.
Ships from and sold by Amazon.com.
Gift-wrap available.

Consider these two user stories that record the functionality associated with this feature:

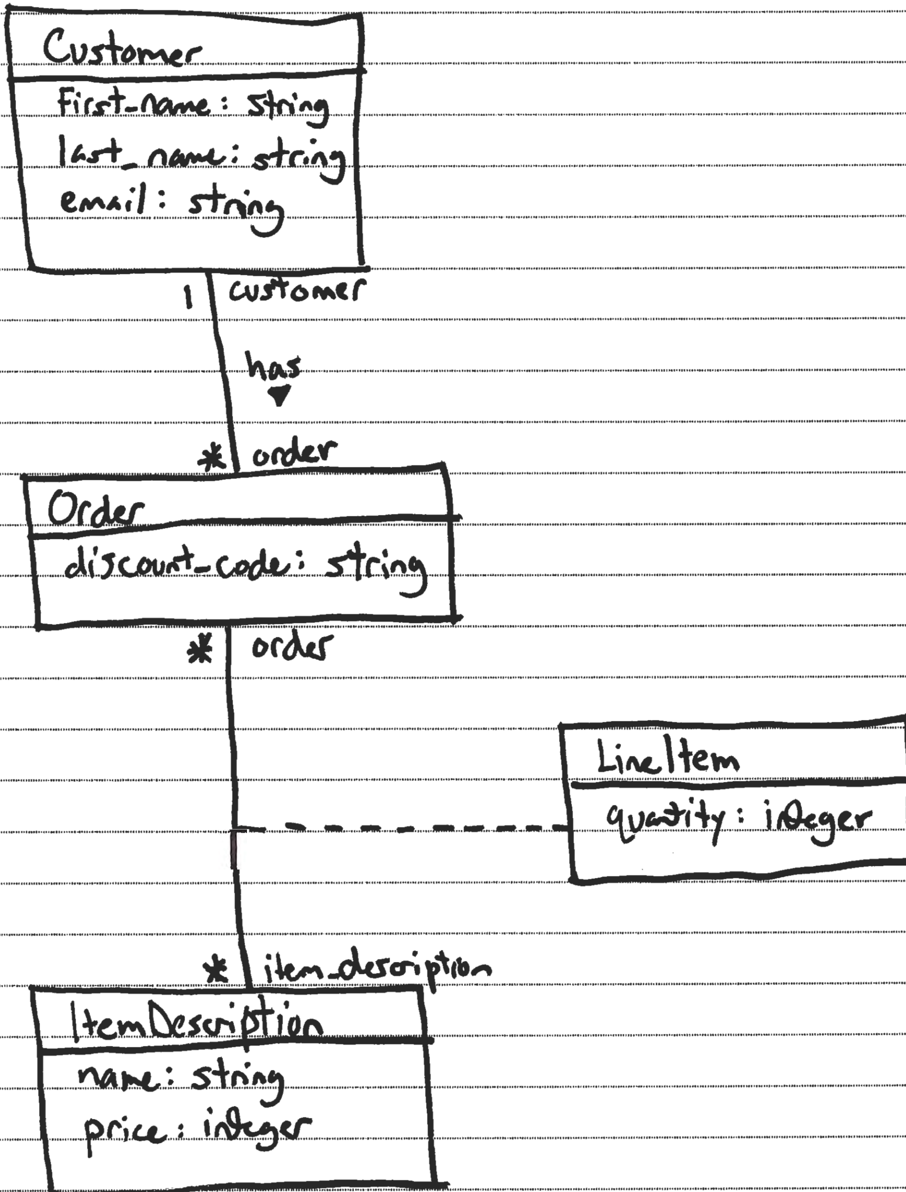
| Title: Add item to cart<br><br>Description: As a shopper, I want to add an item to my cart, so that I can include the item in my order. | Title: Add-to-Cart button<br><br>Description: An "Add to Cart" button will appear on an item-description page. Pressing the button will cause the item to be added to the user's shopping cart. The button must use AJAX/JavaScript to avoid reloading the whole page. |
|---|---|

23. [6%] Which of these to USs is of higher quality? Support your answer by describing two attributes that make one lower quality than the other.

"Add item to cart" is of higher quality.
Attribute: Follows title and description templates.
"Add item to cart" does, and "Add-to-Cart button" doesn't.
Attribute: Not use technical jargon that the customer may not understand.
"Add-to-Cart button" mentions AJAX/JavaScript.
Attribute: Not include UI elements or other implementation/design decisions.
"Add-to-Cart button" mentions pages, buttons, implementation technologies, AJAX & JavaScript.

The questions on the following pages refer to the example figures. The figures show different aspects of the BurgerShop web app for ordering food online. Customers can use the app to place orders, and staff can use the app to view and fulfill orders.

24. [14%] Draw a UML class diagram that represents the four model classes given in Figure 1. Be sure to include all classes, all attributes, all attribute types. Also, label all associations and association ends, and include all multiplicities. Model join-type model classes as association classes in the diagram (as discussed in class). Don't include any "id" attributes (including foreign keys) or any of the "datetime" attributes that Rails provides by default.



25. [8%] Consider the model classes in Figure 1 and the fixtures in Figure 2. Using the lines of code in Figure 3, complete the following model test class such that it has a test for a valid instance of the model class and a test that demonstrates that the model class's attribute validation catches an invalid value. Fill in each blank with the Line ID of the line from the figure that should go there. (The purpose of having you write the Line ID here is to save you the effort of writing the entire lines of code.) You should fill all blanks and use all lines at least once. Some lines may be used more than once. The gaps in each line mark different levels of indentation. You must use these lines/gaps to indent your implementation properly.

```
class LineItemTest < ActiveSupport::TestCase
        test-valid ________ ________ ________ ________
        ________ one= ________ ________ ________
        ________ assert ________ ________ ________
        end ________ ________ ________ ________
        test-quantity ________ ________ ________
        ________ one= ________ ________ ________
        ________ quantity= ________ ________ ________
        ________ assert-not ________ ________
        end ________ ________ ________ ________
end
```

26. [14%] Consider the Developers index page in Figure 4. Using the lines of code in Figure 5, reverse engineer the view code that produced this page. Fill in each blank with the Line ID of the line from the figure that should go there. You should fill all blanks and use all lines at least once. Some lines may be used more than once. The gaps in each line mark different levels of indentation. You must use these lines/gaps to indent your implementation properly.

```
h1
p
    customer:
    name
/p
table
    thead
        tr
            th-item
            th-quantity
            th-price
        /tr
    /thead
    tbody
        each
            tr
                td-item
                td-quantity
                td-sprintf
            /tr
        end
    /tbody
/table
p
    discount8
    discount
/p
p
    edit
    back
/p
```

27. [1%] Which of the following lines of code would the controller need to execute before rendering the form view from Figure 4?

   a. `@order = Order.find(params[:id])`

   b. `@order = Order.new(params.require(:order).permit(:discount_code))`

   c. `@order = Order.new`

   d. `@orders = Order.all`

   e. None of the above

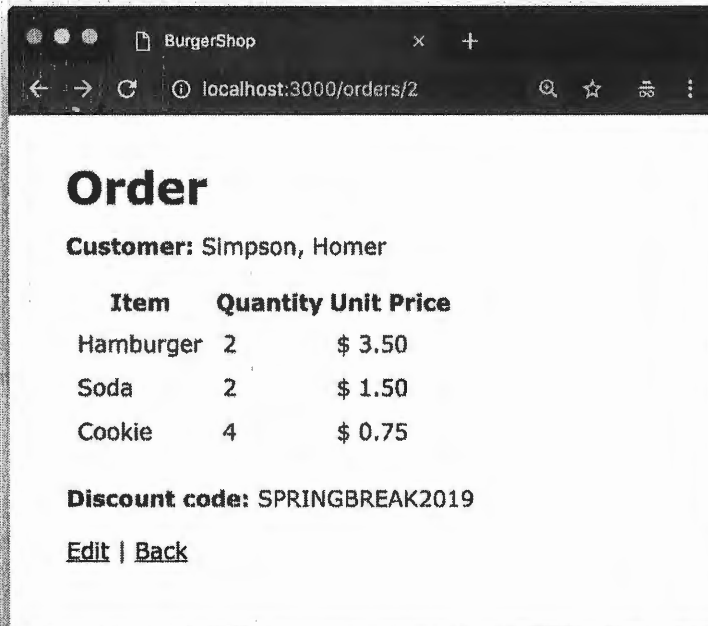



**Figure 4. Order *show* page.**

<table>
<tr><th>Line ID</th><th>Line of Code</th></tr>
<tr><td>each</td><td><% @order.line_items.each do |line_item| %></td></tr>
<tr><td>end</td><td><% end %></td></tr>
<tr><td>name</td><td><%= @order.customer.last_name %>, <%= @order.customer.first_name %></td></tr>
<tr><td>discount</td><td><%= @order.discount_code %></td></tr>
<tr><td>back</td><td><%= link_to 'Back', orders_path %></td></tr>
<tr><td>edit</td><td><%= link_to 'Edit', edit_order_path(@order) %> |</td></tr>
<tr><td>/p</td><td></p></td></tr>
<tr><td>/table</td><td></table></td></tr>
<tr><td>/tbody</td><td></tbody></td></tr>
<tr><td>/thead</td><td></thead></td></tr>
<tr><td>/tr</td><td></tr></td></tr>
<tr><td>h1</td><td><h1>Order</h1></td></tr>
<tr><td>p</td><td><p></td></tr>
<tr><td>customer:</td><td><strong>Customer:</strong></td></tr>
<tr><td>discount:</td><td><strong>Discount code:</strong></td></tr>
<tr><td>table</td><td><table></td></tr>
<tr><td>tbody</td><td><tbody></td></tr>
<tr><td>td-sprintf</td><td><td>$<%= sprintf("%20.2f", line_item.item_description.price/100.0) %></td></td></tr>
<tr><td>td-item</td><td><td><%= line_item.item_description.name %></td></td></tr>
<tr><td>td-quantity</td><td><td><%= line_item.quantity %></td></td></tr>
<tr><td>th-item</td><td><th>Item</th></td></tr>
<tr><td>th-quantity</td><td><th>Quantity</th></td></tr>
<tr><td>th-price</td><td><th>Unit Price</th></td></tr>
<tr><td>thead</td><td><thead></td></tr>
<tr><td>tr</td><td><tr></td></tr>
</table>

**Figure 5. Lines of ERB code for the Order *show* page.**

## Figures

```
# == Schema Information
#
# Table name: customers
#
#  id         :integer          not null, primary key
#  first_name :string
#  last_name  :string
#  email      :string
#  created_at :datetime         not null
#  updated_at :datetime         not null
#

class Customer < ApplicationRecord
  has_many :orders
end
```

```
# == Schema Information
#
# Table name: orders
#
#  id            :integer          not null, primary key
#  discount_code :string
#  created_at    :datetime         not null
#  updated_at    :datetime         not null
#  customer_id   :integer
#

class Order < ApplicationRecord
  belongs_to :customer
  has_many :line_items
  has_many :item_descriptions, through: :line_items
end
```

```
# == Schema Information
#
# Table name: line_items
#
#  id                  :integer          not null, primary key
#  quantity            :integer
#  created_at          :datetime         not null
#  updated_at          :datetime         not null
#  order_id            :integer
#  item_description_id :integer
#

class LineItem < ApplicationRecord
  belongs_to :order
  belongs_to :item_description
  validates :quantity, numericality: { greater_than: 0 }
end
```

```
# == Schema Information
#
# Table name: item_descriptions
#
#  id         :integer          not null, primary key
#  name       :string
#  price      :integer
#  created_at :datetime         not null
#  updated_at :datetime         not null
#

class ItemDescription < ApplicationRecord
  has_many :line_items
  has_many :orders, through: :line_items
end
```

**Figure 1. Four model classes from the BurgerShop app.**

```
one:
  first_name: Homer
  last_name: Simpson
  email: homer@email.com

two:
  first_name: Wilma
  last_name: Flintstone
  email: wflinstone@email.com
```

```
one:
  discount_code: SPRINGBREAK2019
  customer: one

two:
  discount_code: PRESIDENTSDAY2019
  customer: two
```

```
one:
  quantity: 1
  order: one
  item_description: one

two:
  quantity: 2
  order: two
  item_description: two
```

```
one:
  name: Hamburger
  price: 350

two:
  name: Soda
  price: 150
```

**Figure 2. Test fixtures for the BurgerShop model classes.**

| Line ID | Line of Code |
|---|---|
| assert | `assert one.valid?` |
| assert-not | `assert_not one.valid?` |
| end | `end` |
| one= | `one = line_items(:one)` |
| quantity= | `one.quantity = -1` |
| test-valid | `test "line item should be valid" do` |
| test-quantity | `test "quantity must be greater than zero" do` |

**Figure 3. Model unit test lines of code.**