Exam #1 Practice #4 Key

VOCAB KEY

1 Superclass

2 Namespace

3 Version Control

4 Constructor

5 Exception

6 Algorithm

7 Object-Oriented Programming

8 Variable

9 Encapsulation

10 Invariant

MULTIPLE CHOICE KEY

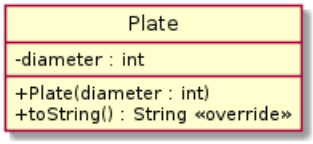
1 B	6 B	11 B
2 C	7 C	12 B
3 B	8 C	13 C
4 B	9 C	14 C
5 D	10 A	15 B

Free Response

Write the solutions to the Free Response questions in the provided space. Additional space is on back, and additional coding sheets are available on request. **Write your name and student ID on EVERY additional coding sheet you use.**

Multi-part questions are based on the class diagrams shown, however, *each question is independent* of the others. Don't ignore the remaining questions if you have trouble answering one, just skip to the next question and continue.

1. {Code a Class} A place setting as shown below consists of 2 plates, a cup, and 3 utensils of cutlery - a knife, a fork, and a spoon. Consider the class diagram. Class `Plate` represents a plate with the diameter (the size of the plate) given in inches.



a. {5 points} In file Plate.java, **begin writing Java class Plate**. For this question, declare the Plate class so that it will be visible to any code in the system, and then define the field. DO NOT write the constructor and method members of class Plate here - they may be written for separate questions below. Assume any required imports - you need not write them.

```
public class Plate {
    private int diameter;
```

b. {6 points} In file Plate.java, assuming all fields are properly declared, **write just the Plate constructor**, properly initializing the field to the parameter value.

```
public Plate(int diameter) {
    this.diameter = diameter;
}
```

c. {6 points} In file Plate.java, assuming all fields and constructors are properly declared, **override the toString method** to convert the object to a String. Ensure that the superclass has a matching toString method to override. For example, an 8" diameter plate would print `8" plate`. (DO NOT return this as a *literal* String, but use the actual field value instead.)

- Any *valid* construction of the String is fine, as long as it actually results in (for example) '8" plate'. For example, `return String.format("%d%c plate", diameter, '"')` would also be fine.

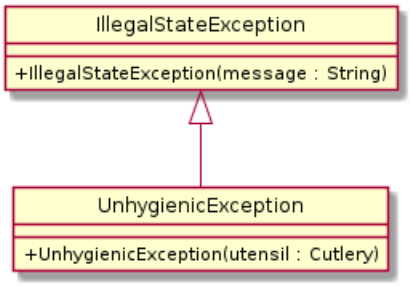
```
@Override
public String toString() {
    return "" + diameter + "\" plate";
}
```

2. {3 points} {Code an enum} Cutlery comes in many ornate forms called *patterns*. **Write the enumeration** in Java of some popular cutlery patterns as shown in the class diagram so it is visible *only* within the current package.

- The requirement is for this to be package-private, thus NO visibility selection is permitted (default for Java).

```
enum Pattern {Rattail, Willow, Kings, Harley}
```

3. {6 points} {Code a custom exception} Unwashed plates, cups, and cutlery are *unhygienic*, or dirty enough to perhaps make you sick. To report the use of unwashed items in a place setting, **create a custom Java exception** `UnhygienicException` as shown in the class diagram. Assume all imports - you do not need to write them. Write the constructor for `UnhygienicException` to chain to its superclass constructor with the String representation of parameter `utensil` followed by "isn't clean!" as its parameter. **Important:** `IllegalStateException` is part of the Java Class Library - DO NOT write it! Write only `UnhygienicException`.

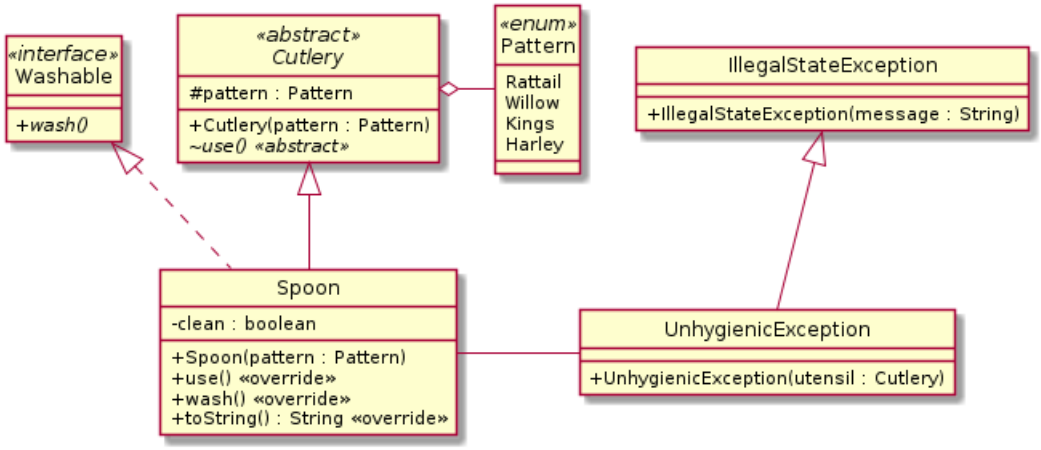



```
class UnhygienicException extends IllegalStateException {
    public UnhygienicException(Cutlery utensil) {
        super(utensil.toString() + " isn't clean!");
    }
}
```

4. {2 points} {Code an interface} Most dirty items can be washed to make them clean again. In file Washable.java, **write interface Washable** as shown in the class diagram.


```
interface Washable {
    void wash();
}
```

5. {Inheritance} Here is a more complete class diagram for cutlery in our place setting (Knife and Fork are omitted).



a. {4 points} Write the name of **any one class member** (NOT enum) in the above diagram that is

- private: **clean**
- protected: **pattern**
- package-private: **use()**
- public: ONE of **wash(), Cutlery, Spoon, use(), toString(), UnhygenicException(), IllegalStateException()**

b. {6 points} In file Cutlery.java, **write abstract class Cutlery**. The constructor should initialize the field to its parameter. Assume any required imports - you need not write them.

- Any implementation provided for abstract method use() is wrong - abstract means "no implementation".

```
abstract class Cutlery {
    protected Pattern pattern;
    public Cutlery(Pattern pattern) {
        this.pattern = pattern;
    }
    abstract void use();
}
```

c. {3 points} In file Spoon.java, **begin writing Java class Spoon**. For this question, declare the Spoon class so that it will be visible to any code in the system and includes the relationships shown, and then define the field. DO NOT write the constructor and method members of class Spoon here - they may be written for separate questions below. Assume any required imports - you need not write them.

```
class Spoon extends Cutlery implements Washable {
    private boolean clean;
```

d. {5 points} In class Spoon, the `clean` field is true if the Spoon object has never been used or has since been washed. In file Spoon.java, **continue writing class Spoon**, writing *only* method `use()`, ensuring that it overrides a superclass or interface method declaration. In method `use()`, if the Spoon is not `clean`, throw an `UnhygienicException` with the current object as the parameter. Otherwise, print "Stirring with" and the object's own String representation to System.out, then set field `clean` to false.

- Alternate, *valid* approaches are acceptable.

```
@Override
public void use() {
    if(!clean) throw new UnhygienicException(this);
    System.out.println("Stirring with " + toString());
    clean = false;
}
```

6. {4 points} {Write a main method} In file Setting.java, assuming all classes and enums above are properly defined in the same package, write static method `main`. In `main`:

- Create an object of type `Spoon` that uses a [pattern may vary] pattern.
- `use` the spoon.
- If an `UnhygienicException` is thrown when the spoon is used, `wash` it and then `use` it again.

```
public static void main(String[] args) {
    Spoon spoon = new Spoon(Pattern.Rattail);
    try {
        spoon.use();
    } catch(UnhygienicException e) {
        spoon.wash();
        spoon.use();
    }
}
```

BONUS #1: {+3 points} Write a valid bash command that would do the following:

- List the files in the current directory: `ls` (also accept `lt` or `exa`)
- Create a new directory named bonus: `mkdir bonus` (also accept `mkcd bonus`)
- Edit file Bonus.java (any editor you choose): `e Bonus.java` (e may be e, gedit, vi, code, or any other editor)
- Add and then commit file Bonus.java to the git repository (using any commit message you choose): `git add Bonus.java ; git commit -m "Bonus!"` (or any message)
- Replace file Bonus.java with the most recently committed version: `git checkout Bonus.java` (but also accepted `git pull` or `git push`.

BONUS #2: {+2 points} In ONE brief sentence, explain why you should always select a License for public GitHub repositories containing code that you intend for others to use.

Without a specified license, anyone using your code is violating your copyright.