Style Guide

Naming Conventions

Use the Java naming conventions

- camelCase for fields, local variables and functions
- UPPER_CASE_WITH_UNDERSCORES for constants
- CapitalizedWords for class names
- allwordslowercase for package names

Descriptive Names

Choose descriptive names. Avoid the following:

- single letter names

```
double t = 10.0; //does t stand for tip? for total?
int n = 2; //does n stand for number?
```

- abbreviations, except where obvious

```
double mpg = 35; //good name!
boolean iaa = true; //bad name! is account active?
```

- meaningless names

```
boolean eatMyShorts = true; //huh?
int puddle = 25; //wait, what?!?!
```

Spacing

Include space for readability. “Let your code breathe!”

Bad Style!

```
public String placeOrder(int id)
{
    //get the order
    Order order = Orders.getById(id);
    if (!order.validate())
    {
        throw new IllegalStateException();
    }
    //calculate the order total
    OrderItem[] items = order.items();
    double sub = 0.0;
    for (OrderItem item : items)
    {
        sub += item.price() * item.num()
    }
    //...
}
```

Good Style!

```
public String placeOrder(int id)
{
    //get the order
    Order order = Orders.getById(id);
    if (!order.validate())
    {
        throw new IllegalStateException();
    }

    //calculate the order total
    OrderItem[] items = order.items();
    double sub = 0.0;

    for (OrderItem item : items)
    {
        sub += item.price() * item.num()
    }

    //...
}
```

Class Formatting

Don't let a line of code stretch off-screen.

- Reduce the length of any line of code so that it fits entirely on-screen in a standard editor.
- This is usually around 80 characters.

Follow the suggested placement for Class elements

```
public class ClassName
{
    //fields

    //constructors

    //methods
}
```

Comments

Include descriptive comments throughout your code. Each logical section of code should have a comment above it.

Bad Style!	Good Style!
`System.out.println("Enter first: ");`	`//get user inputs`
`String fName = scanner.nextLine();`	`System.out.println("Enter first: ");`
`System.out.println("Enter last: ");`	`String fName = scanner.nextLine();`
`String lName = scanner.nextLine();`	`System.out.println("Enter last: ");`
	`String lName = scanner.nextLine();`
`System.out.println(fName + " " + lName);`	
	`//print the results`
	`System.out.println(fName + " " + lName);`

Curly Braces

Same-line style or next-line style are both OK. But pick one style and be consistent. All your braces should use the same format.

Same line!	Next line!
`public Class Point {`	`public Class Point`
`    private int x;`	`{`
`    private int y;`	`    private int x;`
	`    private int y;`
`    public Point(int x, int y) {`	
`        this.x = x;`	`    public Point(int x, int y)`
`        this.y = y;`	`    {`
`    }`	`        this.x = x;`
`}`	`        this.y = y;`
	`    }`
	`}`

Constants

Do not leave unnamed numerical values floating around your code. These are called "magic numbers" and make your code harder to read and maintain. Replace them with named constants. The numbers 0, 1 and 2 are generally not considered magic numbers (unless they have special meaning in your problem set).

Bad style!

```java
Employee[] emps = new Employee[3];
for (int i = 0; i < emps.length; i++)
{
    emps[i] = createEmployee();
}

//did our first employee work overtime?
if (emps[0].getHours() > 40)
{
    System.out.println("Found one!");
}
```

Good style!

```java
final int NUM_EMP = 3;
final int OVERTIME = 40;

Employee[] emps = new Employee[NUM_EMP];
for (int i = 0; i < emps.length; i++)
{
    emps[i] = createEmployee();
}

//did our first employee work overtime?
if (emps[0].getHours() > OVERTIME)
{
    System.out.println("Found one!");
}
```

Redundancy

Don't ever, ever, ever repeat yourself in code. This is one of the core skills of a seasoned software developer. Use loops and functions to remove redundancy in your code.

Bad style!

```java
Employee[] employees = getEmployees();

Employee first  = employees[0];
System.out.println(first.getName());
System.out.println(first.position());
first.giveRaise(0.10);

Employee second = employees[1];
System.out.println(second.getName());
System.out.println(second.position());
second.giveRaise(0.10);

Employee third = employees[2];
System.out.println(third.getName());
System.out.println(third.position());
third.giveRaise(0.10);
```

Good style!

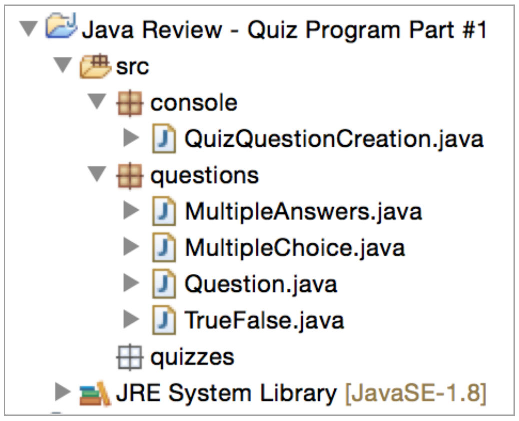
```java
Employee[] employees = getEmployees();

for (int i = 0; i < employees.length; i++)
{
    Employee cur = employees[i];
    System.out.println(cur.getName());
    System.out.println(cur.position());
    cur.giveRaise(0.10);
}
```

Note: it can sometimes be very challenging removing redundancies in your programs. With more practice, removing repeated code becomes easier

Packages

Never leave classes in the default package. Create a new package to group together your classes under a namespace. The name of your package should follow the "reverse internet domain" style.



```
Java Review - Quiz Program Part #1
  src
    console
      QuizQuestionCreation.java
    questions
      MultipleAnswers.java
      MultipleChoice.java
      Question.java
      TrueFalse.java
    quizzes
  JRE System Library [JavaSE-1.8]
```

Formal Documentation – Comment Header

Every class should have a multi-line comment header at the very top of the document. The comment should include:

- The author's name
- The date
- The filename
- A brief description

Example:

```
/*
 * Bob Smith
 * 9/19/2015
 * my_filename.java
 * This file contains a class with some Java code!
 */
```

Formal Documentation – Javadocs

Each class should have full Javadocs. Your file should include:

- A class-level Javadoc
- A Javadoc for each public method

Example:

```
/**
 * This class represents a person
 *
 * @author Sarah Smithers
 * @version 1.0
 */
public class Person
{
    private String name;
    private String nickname;


    /**
     * Creates a new Person with a name and nickname.
     *
     * @param name the name of the new person
     * @param nickname the nickname of the new person
     */
    public Person(String name, String nickname)
    {
        this.name = name;
        this.nickname = nickname;
    }

    /**
     * Gets the full name of the person.
     * @return the name and nickname of the person
     */
    public String getFullName()
    {
        return name + "(" + nickname + ")";
    }

    private void printName()
    {
        System.out.println(getFullName());
    }
}
```