Fundamentals of Java

Design

"this" keyword

An object may refer to itself using the keyword "this":

```
this.method()
this.field
```

Usually, this is equivalent to:

```
method()
field
```

Method overloading

Rule 1: Two methods can have the same name if they have a different number of parameters:

```
public void foo(int x, int y)
public void foo(int x, int y, int z)
```

Rule 2: Two methods can have the same name if they have different parameter ypes:

```
public void bar(double x)
public void bar(int x)
```

Method overloading - Example 1

```
public double average(double x, double y) {
    return (x + y) / 2.0)
}
public double average(double x, double y, double z) {
    return (x + y + z) / 3.0;
}

System.out.println(average(2.0, 3.0));       // prints 2.5
System.out.println(average(7.0, 8.0, 9.0)); // prints 8.0
```

Method overloading - Example 2

```
public void formatReport(String name, int age, String gender)
{
    System.out.println("Name:        " + name);
    System.out.println("Age:         " + age);
    System.out.println("Gender:      " + gender);
}

public void formatReport(String name, int age, String gender, String occupation)
{
    System.out.println("Name:        " + name);
    System.out.println("Age:         " + age);
    System.out.println("Gender:      " + gender);
    System.out.println("Occupation:  " + occupation);
}
```

Constructor overloading

Constructors can also be overloaded:

```
public Person()
public Person(String name)
public Person(int age)
public Person(String name, int age)
```

Constructor overloading - Example

```
public class Person {
	private String name;
	private int age;
	private String occupation;
	public Person(String name, int age) {
		this.name = name;
		this.age = age;
		this.occupation = "none";
	}
	public Person(String name, int age, String occupation) {
		this.name = name;
		this.age = age;
		this.occupation = occupation;
	}
}
```

Code reuse

Badly written programs often contain repeated code. This creates the following problems:

- Programs are larger than they need to be (and therefore harder to read)
- A bug fix or feature enhancement may now need to be repeated in several places.

Repeated code can be eliminated through code reuse.

Code reuse in overloaded methods

```
public void formatReport(String name, int age, String gender) {
    System.out.println("Name:        " + name);
    System.out.println("Age:         " + age);
    System.out.println("Gender:      " + gender);
}
public void formatReport(String name, int age, String gender, String occupation) {
    System.out.println("Name:        " + name);
    System.out.println("Age:         " + age);
    System.out.println("Gender:      " + gender);
    System.out.println("Occupation:  " + occupation);
}
```

The second method repeats the same first three lines of code as the first method.

Code reuse in overloaded methods Solution

```
public void formatReport(String name, int age, String gender) {
    System.out.println("Name:        " + name);
    System.out.println("Age:         " + age);
    System.out.println("Gender:      " + gender);
}

public void formatReport(String name, int age, String gender, String occupation) {
    formatReport(name, age, gender);
    System.out.println("Occupation: " + occupation);
}
```

The solution is for the second method to "reuse" the first method to do the work of printing the first 3 lines of the report.

Code reuse in overloaded constructors

```
public class Person {
    private String name;
    private int age;
    private String occupation;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        this.occupation = "none";
    }
    public Person(String name, int age, String occupation) {
        this.name = name;
        this.age = age;
        this.occupation = occupation;
    }
}
```

The second constructor repeats the same first two lines found in the first constructor.

Code reuse in overloaded constructors - Solution #1

```
public class Person {
    private String name;
    private int age;
    private String occupation;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        this.occupation = "none";
    }
    public Person(String name, int age, String occupation) {
        this(name, age);
        this.occupation = occupation;
    }
}
```

The statement this(name, age); invokes the first constructor. This reuses the existing code to handle the name and age.

Code reuse in overloaded constructors - Solution #2

```
public class Person {
    private String name;
    private int age;
    private String occupation;
    public Person(String name, int age) {
        this(name, age "none");
    }
    public Person(String name, int age, String occupation) {
        this.name = name;
        this.age = age;
        this.occupation = occupation;
    }
}
```

The previous solution reused only the code for name and age. The new version reuses the code for name, age and occupation.

Code reuse with method parameters

Two blocks of similar code can be refactored into reusable blocks via the use of method parameters.

1. Identify the similarities between two blocks of code.
2. Identify the differences between two blocks of code.
3. Create the reusable method based on these observations.
4. Use the reusable method.

Code reuse with method parameters Step 1

Step 1. Identify code that has been repeated.

```
public void myMethod()
{
    System.out.println("Hello " + player1.name());
    System.out.println("Your score is " + player1.score());
    System.out.println("Well done!");
    ------------------------------
    System.out.println("Hello " + player2.name());
    System.out.println("Your score is " + player2.score());
    System.out.println("Well done!");
}
```

Lines 4-6 repeat lines 1-3
(with some minor differences).

Code reuse with method parameters Step 2

Step 2. Identify the differences:

```
public void myMethod()
{
    System.out.println("Hello " + player1.name());
    System.out.println("Your score is " + player1.score());
    System.out.println("Well done!");
    ------------------------------
    System.out.println("Hello " + player2.name());
    System.out.println("Your score is " + player2.score());
    System.out.println("Well done!");
}
```

Code reuse with method parameters Step 3

Step 3. Take one copy of the repeated code, and place it into a **new method**:

```
private void printPlayerScore(Player player)
{
    System.out.println("Hello " + player.name());
    System.out.println("Your score is " + player.score());
    System.out.println("Well done!");
}
```

The *differences* become *parameters*.
i.e. player1/player2 becomes
the parameter "player".

Code reuse with method parameters - Step 4

Step 4. Delete all repeated code from myMethod() and instead reuse the printPlayerScore() method twice:

```
public void myMethod()
{
    printPlayerScore(player1);
    printPlayerScore(player2);
}

private void printPlayerScore(Player player)
{
    System.out.println("Hello " + player.name());
    System.out.println("Your score is " + player.score());
    System.out.println("Well done!");
}
```