Chad Philip Johnson
CSCI20, Worthington
October 13th, 2012

Pencil & Paper 10

1. What is method overloading?
Method overloading allows two or more methods to have the same name, provided each contains a different set of arguments (or, more specifically, a unique signature). According to our textbook, an overloaded method has "two (or more) definitions of a single method name…. When you overload a method name, any two definitions of the method name must have different signatures; that is, any two definitions of the method name either must have different numbers of parameters or some parameter position must be of differing types in the two definitions" (221).

2. How does Java distinguish between different versions of an overloaded method?
Each overloaded method must have a different method signature than the other methods which share the same name For example: myMethod(int var1, string var2, double var3) has a method signature of myMethod(int, string, double), whereas myMethod(string var1, int var2, double var3) has the different method signature of myMethod(string, int, double). Java is able to distinguish between the two overloaded forms of myMethod based on these different signatures.

3. What is a method's *signature*?
According to our textbook, a method signature "consists of the method name and the list of types for parameters that are listed in the heading of the method name" (221).

4. Can a class define more than one method with the exact same signature?
No, a class cannot define more than one method with the exact same signature. According to our textbook, "When you overload a method name, each of the method definitions in the class must have a different signature" (221). The method name and the types and order of its arguments represent the method's signature.

5. Can you overload constructors?
Yes, overloading of constructors is allowed in Java. The desired overloaded constructor can be invoked by passing arguments to the class during instantiation. For example: Shoe myShoe = new Shoe("Reebok", "Rubber", 185.00, 1);

6. Should method overloading be used instead of creating different methods with descriptive names? Briefly explain your answer.
When the possibility of automatic typecasting by Java is introduced to an overloaded method, it is better to use different methods with descriptive names. For example, if the overloaded method myMethod has the two signatures myMethod(int) and myMethod(double) and it is invoked with myMethod(2), the first form is used because Java matches the integer 2 with the signature which specifies an integer for its argument. Calling the form with the signature for a double requires explicit typecasting of the number 2 and/or passing numbers of type float or double that will match this specific signature--in this case, Java would automatically typecast a float into a double, so long as there wasn't an overloaded form of the method with the signature myMethod(float).

Our textbook provides a more involved example where the overloaded function doSomething has the two signatures doSomething(double, int) and doSomething(int,

double). In this case a method invocation with non-specific arguments such as doSomething(5, 10) produces an error because Java is unable to make a decision between the two forms of the method (223). In this case, creating a separate method with a more descriptive name would prevent potential errors. It also has the added benefit of making the program more understandable to others.

7. Can you overload a method simply by changing the return type of a method?
No, an overloaded method cannot be created by simply changing the return type. According to our textbook, "The signature of a method lists only the method name and the types of the parameters and does not include the type returned…. The type returned has nothing to do with the signature of a method" (224).

8. List the methods in the class *Shoe* below that are NOT overloaded.
The following methods in the provided code example are not overloaded:
public void setModel(String newModel)
public void setMaterial(String newMaterial)
public void setPrice(double newPrice)
public void setYearsWarranty(int newYearsWarranty)
public String getModel()
public String getMaterial()
public double getPrice()
public int getYearsWarranty()

9. List the methods in the class Shoe below that are overloaded.
The following methods in the provided code example are overloaded:
public Shoe()
public Shoe(String newModel, String newMaterial, double newPrice, int newYearsWarranty)
public void set(String newModel, String newMaterial)
public void set(double newPrice, int newYearsWarranty)
public void set(String newModel, String newMaterial, double newPrice, int newYearsWarranty)
public void display()
public void display(Boolean withLabels)

10. Write the code to (1) create an instance of *Shoe* named *someShoe* using the default *Shoe* constructor, AND (2) create another instance of *Shoe* named *myShoe* using the overloaded constructor and the values "Chucks" (model), "Canvas" (material), 50.0 (price), and 1 (yearsWarranty).
The following code performs the requested tasks:
Shoe someShoe = new Shoe();
Shoe myShoe = new Shoe("Chucks", "Canvas", 50.0, 1);