CSA WS 11G (2 of 2)

Started: Feb 26 at 11:11am

Quiz Instructions

Question 1

100 pts

Consider the following code snippet:

```
if(anObject instanceof Auto)
{
   Auto anAuto = (Auto)anObject;
   . . .
}
```

What does this code do?

- This code tests whether `anObject` was created from a superclass of `Auto`.
- This code creates a subclass type object from a superclass type object.
- This class safely converts an object of any type to an object of type `Auto`.
- (selected) This code safely converts an object of type `Auto` or a subclass of `Auto` to an object of type `Auto`.

Question 2

100 pts

Which of the following statements about abstract methods is true?

- (●) An abstract method has a name, parameters, and a return type, but no code in the body of the method.
- ( ) An abstract method has parameters, a return type, and code in its body, but has no defined name.
- ( ) An abstract method has a name, a return type, and code in its body, but has no parameters.
- ( ) An abstract method has only a name and a return type, but no parameters or code in its body.

Question 3

100 pts

Which of the following statements about inheritance is correct?

- ( ) You can always use a superclass object in place of a subclass object.
- (●) You can always use a subclass object in place of a superclass object.
- ( ) A superclass inherits data and behavior from a subclass.
- ( ) A superclass inherits only behavior from a subclass.

Question 4

100 pts

Which of the following is true regarding subclasses?

- A subclass that inherits methods from its superclass may not override the methods.
- A subclass that inherits instance variables from its superclass may not declare additional instance variables.
- A subclass may inherit methods or instance variables from its superclass but not both.
- (selected) A subclass may inherit methods and instance variables from its superclass, and may also implement its own methods and declare its own instance variables.

Question 5

100 pts

Consider the classes shown below:

```
public class Parent
{
   private int value = 100;
   public int getValue()
   {
      return value;
   }
}

public class Child extends Parent
{
   private int value;
   public Child(int number)
   {
      value = number;
   }
```

```
}
```

What is the output of the following lines of code?

```
Child kid = new Child(-14);
Parent adult = new Parent();
System.out.println(kid.getValue() + " " + adult.getValue());
```

- (●) `100 100`
- ( ) `-14 100`
- ( ) `-14 -14`
- ( ) `100 -14`

Question 6

100 pts

Consider the following class hierarchy:

```
public class Vehicle
{
   private String type;
   public Vehicle(String type)
   {
      this.type = type;
   }
   public String getType()
   {
      return type;
```

```
    }
}

public class LandVehicle extends Vehicle
{
   public LandVehicle(String type)
   {
      . . .
   }
}

public class Auto extends LandVehicle
{
   public Auto(String type)
   {
      . . .
   }
}
```

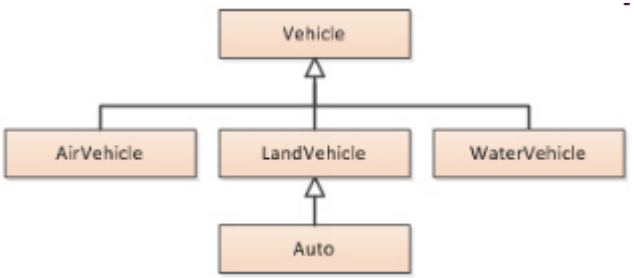
Which of the following code fragments is NOT valid in Java?

- `Vehicle myAuto = new Auto("sedan");`
- `LandVehicle myAuto = new Auto("sedan");`
- `Auto myAuto = new Auto("sedan");`
- (selected) `LandVehicle myAuto = new Vehicle("sedan");`

Question 7

100 pts

Consider the following inheritance hierarchy diagram:



Cite the relationship between the `Auto` class and other class(es).

- ○ `Auto` is a superclass of `LandVehicle`; and `LandVehicle` is a superclass of `Vehicle`.
- ○ `Auto` is a superclass of `LandVehicle`; and `LandVehicle` is a subclass of `Vehicle`.
- ○ `Auto` is a subclass of `LandVehicle`; and `LandVehicle` is a superclass of `Vehicle`.
- ◉ `Auto` is a subclass of `LandVehicle`; and `LandVehicle` is a subclass of `Vehicle`.

Question 8

100 pts

Consider the classes shown below:

```
public class Parent
{
   public void doSomething()       // method 1
   { /* Implementation not shown */ }
}
```

```
public class Child extends Parent
{
   public void doSomething(int n)    // method 2
   { /* Implementation not shown */ }

   public void doSomething()         // method 3
   { /* Implementation not shown */ }
}
```

In the following code segment,, which version of the `doSomething` method can be called on the variable `kid`?

```
Parent kid = new Child();
kid._____;
```

- (●) Method 1 only
- ( ) Methods 2 and 3 only
- ( ) Methods 1 and 2 only
- ( ) Methods 1, 2, and 3
- ( ) Method 3 only

Question 9

100 pts

Consider the following class hierarchy:

```
public final class Shape
{
```

```
   private String mycolor;
   public Shape(String mycolor)
   {
      this.type = mycolor;
   }
   public String getColor()
   {
      return mycolor;
   }
}

public class Triangle extends Shape
{
   public Triangle(String mycolor)
   {
      super(mycolor);
   }
}
```

What is wrong with this class hierarchy definition?

- ○ Nothing is wrong with the code.
- ○ There should be more subclasses of the `Shape` class than just `Triangle`.
- ◉ There cannot be any subclasses of the `Shape` class.
- ○ It is not possible to use `super` in the `Triangle` constructor.

Question 10

100 pts

Suppose the class `Message` is partially defined as shown below:

```
public class Message
{
   private String value;

   public Message(String initial)
   {
      value = initial;
   }

   public String getMessage()
   {
      return value;
   }
}
```

A subclass of Message, `ExcitedMessage`, is defined that will behave like `Message`, except that it will add two exclamation points to the end of the message. Sample code that uses `ExcitedMessage` is shown below.

```
ExcitedMessage greeting = new ExcitedMessage("Hello");
System.out.print(greeting.getMessage());//prints "Hello!!"
```

Which `ExcitedMessage` constructor will give this behavior?

- (●)
```
public ExcitedMessage(String line)
   {
      super(line + "!!");
   }
```

- ( )
```
public ExcitedMessage(String line)
```

```
    {
       value = line + "!!";
    }
```

```
public ExcitedMessage(String line)
    {
       line = line + "!!";
       super(line);
    }
```

```
public ExcitedMessage(String line)
    {
       new Message(line + "!!");
    }
```

Question 11

100 pts

Consider the following code snippet:

```
public class Vehicle
{
   private String manufacturer;
   . . .
   public void setVehicleClass(double numberAxles)
   {
       . . .
   }
}
```

If a `Motorcycle` class is created as a subclass of the `Vehicle` class, which of the following statements is correct?

- A `Motorcycle` object inherits and can directly use both the instance variable `manufacturer` and the method `setVehicleClass`.
- A `Motorcycle` object inherits and can directly use the instance variable `manufacturer` but not the method `setVehicleClass`.
- A `Motorcycle` object inherits but cannot directly use either the instance variable `manufacturer` or the method `setVehicleClass`.
- (selected) A `Motorcycle` object inherits and can directly use the method `setVehicleClass` but cannot directly use the instance variable `manufacturer`.

Question 12

100 pts

Consider the following code snippet:

```
public class Score
{
   private String name;
   . . .
   public boolean equals(Object otherScore)
   {
      Score other = (Score) otherscore;
      return name.equals(other.name);
   }
   . . .
}
```

What is wrong with this code?

- The `return` statement should use the `==` operator instead of the `equals` method.

- The parameter in the `equals` method should be declared as `Score otherScore`.
- It is an error to cast `otherScore` as a `Score` object before using the `equals` method.
- (•) There is nothing wrong with this code.

Question 13

100 pts

Consider the classes shown below:

```
public class Parent
{
   private int value = 100;
   public int getValue()
   {
      return value;
   }
}

public class Child extends Parent
{
   private int value;
   public Child(int number)
   {
      value = number;
   }
   public int getValue()
   {
```

```
        return value;
    }
}
```

What is the output of the following lines of code?

```
Child kid1 = new Child(-14);
Child kid2 = new Child(21);
System.out.println(kid1.getValue() + " " + kid2.getValue());
```

- (●) -14 21
- ( ) 21 21
- ( ) 21 100
- ( ) 100 100

Question 14

100 pts

What must a subclass do to modify a private superclass instance variable?

- ( ) The subclass must simply use the name of the superclass instance variable.
- ( ) The subclass must declare its own instance variable with the same name as the superclass instance variable.
- (●) The subclass must use a public method of the superclass (if it exists) to update the superclass's private instance variable.
- ( ) The subclass must have its own public method to update the superclass's private instance variable.

Question 15

100 pts

Which of the following statements is true about using the reserved word `super` to call a superclass constructor?

- ○ The call must use the keyword `super` followed by a period and a method name.
- ○ The call must use the keyword `super` with no arguments.
- ○ The call must be the last line of the subclass constructor.
- ◉ The call must be the first line of the subclass constructor.

Question 16

100 pts

To ensure that an instance variable can only be accessed by the class that declared it, how should the variable be declared?

- ○ `public`
- ◉ `private`
- ○ `protected`
- ○ `final`

Question 17

100 pts

Consider the following code snippet:

```
Vehicle aVehicle = new Auto();
aVehicle.moveForward(200);
```

Assume that the `Auto` class inherits from the `Vehicle` class, and both classes have an implementation of the `moveForward` method with the same set of parameters and the same return type. The process for determining which class's `moveForward` method to execute is called ____.

- inheritance disambiguation.
- inheritance hierarchy.
- dynamic inheritance.
- dynamic method lookup.

Question 18

100 pts

Consider the following code snippet:

```
Employee anEmployee = new Programmer();
anEmployee.increaseSalary(2500);
```

If the `Programmer` class inherits from the `Employee` class, and only the `Employee` class has an implementation of the `increaseSalary` method, which statement is correct?

- The `increaseSalary` method call will cause a run-time error.

- (●) The `increaseSalary` method of the `Employee` class will be executed.
- ( ) The `Programmer` class is required to provide an implementation of the `increaseSalary`method.
- ( ) `Programmer` objects must be cast to `Employee` objects before the method call can me made.

Question 19

100 pts

Consider the following code snippet:

```
Employee programmer = new Employee(10254, "exempt");
String s = programmer.toString();
```

Assume that the `Employee` class has not implemented its own `toString()` method. What value will `s` contain when this code is executed?

- ( ) `s` will contain the values of the instance variables in `programmer`.
- ( ) `s` will contain only the class name of the `programmer` object.
- (●) `s` will contain the class name of the `programmer` object followed by a hash code.
- ( ) This code will not compile.

Question 20

100 pts

Consider the following code snippet:

```
public class Coin
```

```
{
   private String name;
   . . .
   public boolean equals(Object otherCoin)
   {
      return name.equals(otherCoin.name);
   }
   . . .
}
```

What is wrong with this code?

- ○ The `return` statement should use the `==` operator instead of the `equals` method.
- ○ The parameter in the `equals` method should be declared as `Coin otherCoin`.
- ◉ `otherCoin` must be cast as a `Coin` object before using the `equals` method.
- ○ There is nothing wrong with this code.

Question 21

100 pts

Consider the following code snippet:

```
public class Vehicle
{
   . . .
   public void setVehicleClass(double numberAxles)
   {
```

```
        . . .
    }
}

public class Auto extends Vehicle
{
    . . .
    public void setVehicleClass(int numberAxles)
    {
        . . .
    }
}
```

Which of the following statements is correct?

- (●) The `Auto` class overrides the `setVehicleClass` method.
- ( ) The `Vehicle` class overrides the `setVehicleClass` method.
- ( ) The `Auto` class overloads the `setVehicleClass` method.
- ( ) The `Vehicle` class overloads the `setVehicleClass` method.

Question 22

100 pts

Which of the following statements about comparing objects is correct?

- ( ) The purpose of the `equals` method is to compare whether two references are to the same object.

- (●) The purpose of the `equals` method is to compare whether two objects have the same contents.
- ( ) The `==` operator is used to compare whether two objects have the same contents.
- ( ) For objects other than `Object`, the `equals` method and the `==` operator always perform the same actions.

Question 23

100 pts

Consider the following code snippet:

```
Vehicle aVehicle = new Auto(4,"gasoline");
String s = aVehicle.toString();
```

Assume that the `Auto` class inherits from the `Vehicle` class, and neither class has an implementation of the `toString()` method. Which of the following statements is correct?

- (●) The `toString()` method of the `Object` class will be used when this code is executed.
- ( ) The `toString()` method of the `String` class will be used when this code is executed.
- ( ) This code will not compile because there is no `toString()` method in the `Vehicle` class.
- ( ) This code will not compile because there is no `toString()` method in the `Auto` class.

Question 24

100 pts

Consider the following code snippet:

```
public abstract class Machine
```

```
{
   public abstract void setRPMs();
   . . .
}
```

You wish to create a concrete subclass named `PolisherMachine`. Which of the following is the correct way to declare this subclass?

- ```
  public class PolisherMachine implements Machine
  {
  public void setRPMs() { . . . }
  }
  ```

- ```
  public class PolisherMachine extends Machine
  {
  void setRPMs() { . . .}
  }
  ```

- ```
  public class PolisherMachine implements Machine
  {
  void setRPMs() { . . .}
  }
  ```

- ```
  public class PolisherMachine extends Machine
  {
  public void setRPMs() { . . . }
  }
  ```

Question 25

100 pts

Which of the following is true regarding inheritance?

- ○ When creating a subclass, all methods of the superclass must be overridden.
- ○ When creating a subclass, no methods of a superclass can be overridden.
- ◉ A superclass can force a programmer to override a method in any subclass created from it.
- ○ A superclass cannot prevent a programmer from overriding a method in any subclass created from it.

Quiz saved at 9:09pm

Submit Quiz