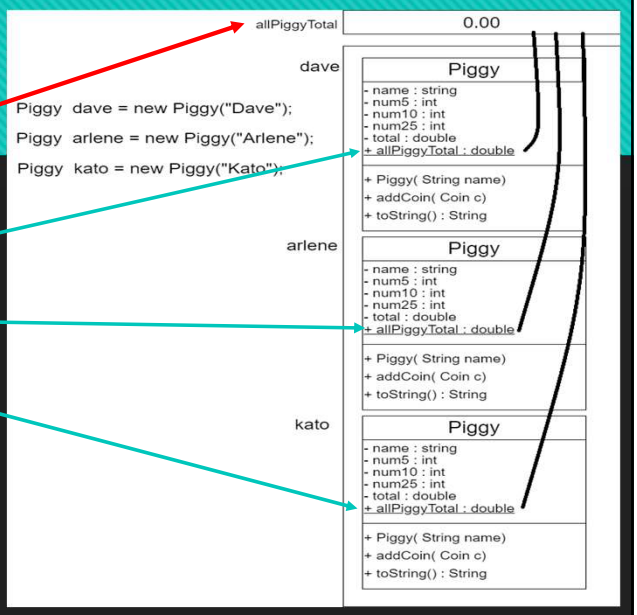
V107 A Review of previous days most important points.

80%

Static Class Variables



allPiggyTotal is a static class variable

- it's underlined in UML diagrams

- there's just one memory location assigned to the static variable

- All instances have a reference to that one memory variable.

allpiggyTotal is the sum of the total amount contained in all Piggy bank instances

78%

Overloaded Methods

Overloading is creating a method that has the same name as another method in the class (or a parent class), but with different parameters and/or return value

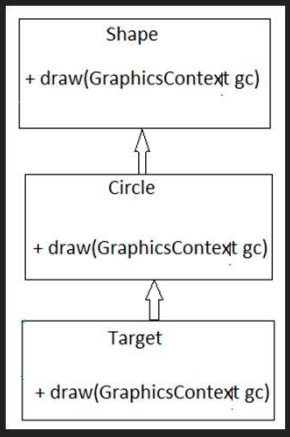
```
public double getMin( double n1, double n2 ) {
    if (n1 < n2) return n1;
    return n2;
}

public int getMin ( int n1,  int n2 ) {
    if (n1 < n2) return n1;
    return n2;
}
```

53%

Overrided Methods

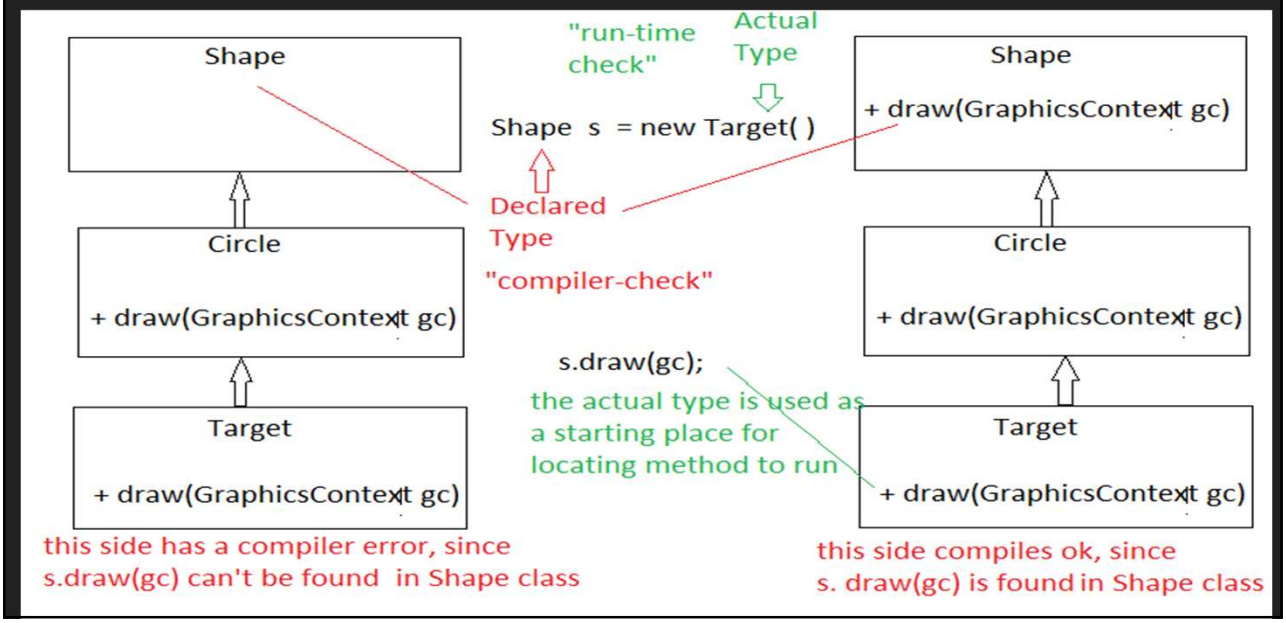
Overrided methods are created when a sub (child) class that has the same name, return value, and parameters as a method in one of the super (parent) classes.

The draw method is overloaded twice in the sketch to the right.



Polymorphism (tricky)

The method .draw(gc) below has a declared type of Shape and an actual type of Target. The compiler checks if the declared type has a .draw(gc) method and if not generates a compiler error. The actual type, i.e. Target has a .draw(gc) method, so at run-time, this method is executed, BUT if Target didn't have a .draw(gc) method, it would search up the inheritance tree for a .draw(gc) method.



67%

A5. Write a loop to initialize all 50 elements of the array below to: true

boolean[ ] b = new boolean[50];

//standard for
for (int i=0; i< 50; i++)
b[i] = true;

```
//Enhanced Loop – can't set array elements using
//the temporary x variable; five "false" values will occur
public class TestProgram
{
   public static void main(String[] args) {

      boolean[] b = new boolean[50];

      for(boolean x : b)
        x = true;

      for(int i=0; i< b.length; i++)
        System.out.println(b[i]);
   }
}
```

X

68%

Enhanced For loop Limitations

```
public static void main(String[] args)
  {
     Die [] dice = new Die [1000000];

     for (Die d : dice) {
        d = new Die();
  }
```

The code above does not create Die objects in each array element because of how the enhanced for loop works in Java.
In the enhanced for loop, the variable d is not a reference to the actual array element, but rather a temporary variable that holds the value of each element in the array during each iteration.
Assigning a new value to d does not modify the corresponding array element.

The code is assigning a new Die object to the temporary variable **d**, but it does not update the array element dice[i]. As a result, all elements of the dice array remain null.

To correctly initialize each array element with a Die object, you should use a regular for loop instead, like this:

```
public static void main(String[] args) {
    Die[] dice = new Die[1000000];

    for (int i = 0; i < dice.length; i++) {
        dice[i] = new Die();
    }
}
```

Why is Inheritance Good

Inheritance in Java provides a powerful mechanism for code reuse, flexibility, and organization, making it a fundamental concept in object-oriented programming.

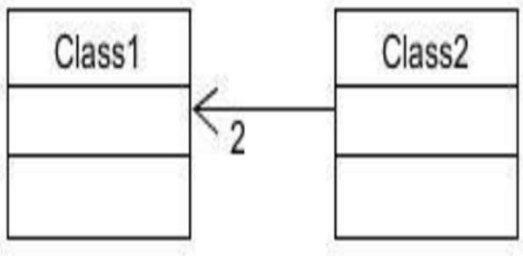
1. Reduces duplicate code.
2. By extending a parent class, you inherit methods that you don't have to re-write.
3. Overriding: allows you to provide specific tailored methods for a class.
4. Polymorphism: objects of different classes can be treated uniformly through a common interface.
5. Maintenance: easier to maintain the code, everything in just one spot.
6. Allows you to extend the functionality of existing classes by adding new features in the child classes.

47%

Association - refers to a "has a" relationship between 2 objects. That is, "Class2 has a Class1"

Write the minimum amount of code necessary to represent the two classes in the UML class diagram on the right.

```
public Class1 {}
public Class2 {
    Class1: a, b;
}
```



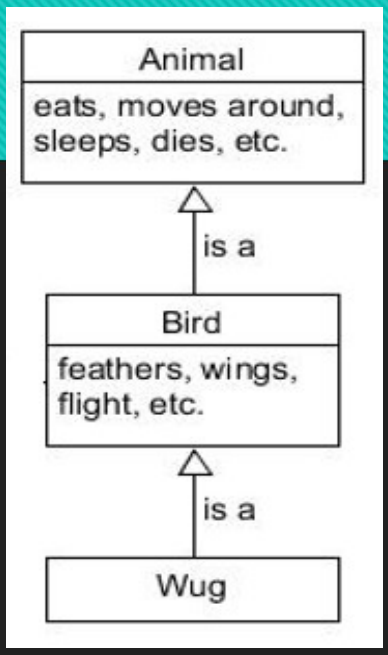
65%

Inheritance - "is a" relation

(a class can only extend to one parent)

A Wug is a Bird. A Bird is a Animal

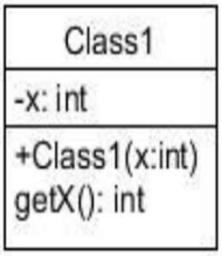
```
public class Bird extends Animal {

}

public class Wug extends Bird {

}
```



67%

Array of Objects

Write the minimum amount of Java code necessary to create an array of 10 objects of type Class1 (see the diagram on the right). Each Class1 object in the array should have a different value for its x variable.




```
Class1[] a = new Class1[10];
for (int i = 0; i < a.length; i++)
    a[i] = new Class1(i);
```

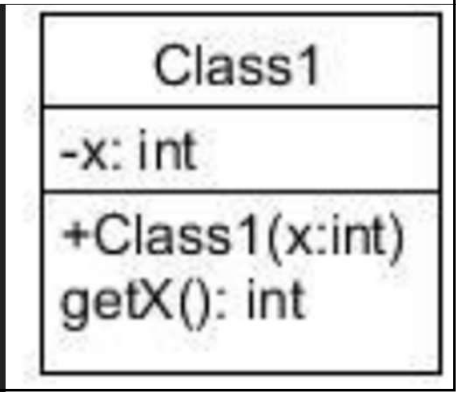
creates an empty array of all, null

each element becomes a Class1 inside the loop

Array of Objects

Write a single line of Java code to print the x value of the 3rd item of the array called, a

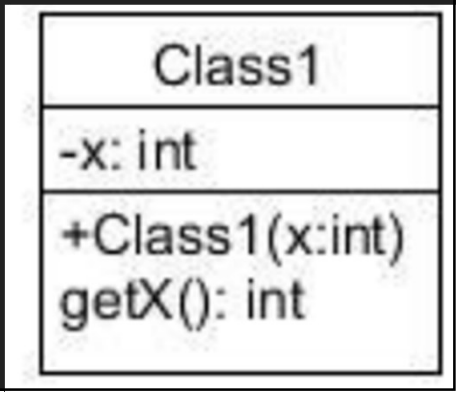
```
System.out.println(  a[2].getX() )
```



Array of Objects

Write an enhanced for loop to display all of the "x" values from each element of array, a

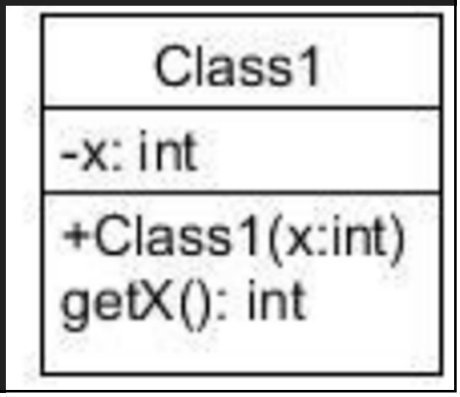
```
for ( Class1  c :  a )  {

    System.out.println( c.getX() );

}
```



Array of Objects

Write an standard for loop to display all of the "x" values from each element of array, a

```
for ( int i=0; i< a.length ; i++)  {

    System.out.println( a[i].getX() );

}
```



66%

GUI Programming

Suppose you have a Button called, button.
What is the trigger line in the GUI program which will activate a handler routine to process the button's purpose?

```
// 5. Add Event Handlers and do final setup quarter
        button.setOnAction ( this::buttonHandler );
```

44%

GUI Programming

What is the method's return type, name and signature if the If the user presses the Button called, button?

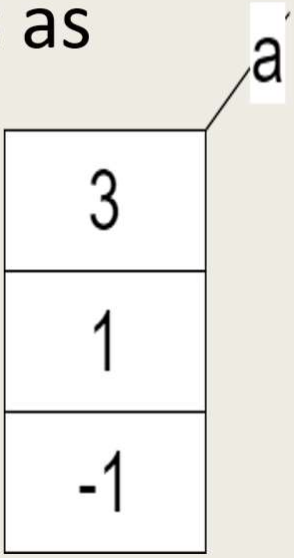
```
button.setOnAction ( this::buttonHandler );
```

```
public void buttonHandler(ActionEvent e) {

}
```

87%

A10. Instantiate and populate array, "a" in as few java code lines as possible.

```
int [ ] a = { 3, 1, -1 };
```



Scanner Class (special considerations)

In this code, after entering an integer and pressing Enter, if you try to enter a string, the nextLine() method won't behave as expected.

It will immediately consume the newline character left in the buffer by nextInt(), resulting in an empty string.

```
Scanner keybd = new Scanner(System.in);

System.out.print("Enter an integer: ");
int x = keybd.nextInt();

System.out.print("Enter a string: ");
String str = keybd.nextLine(); // This line might not work as expected
```

To address this issue, it's a common practice to add an extra keybd.nextLine() after reading an integer or double to consume the remaining newline character before reading the next line.

```
Scanner keybd = new Scanner(System.in);

System.out.print("Enter an integer: ");
int x = keybd.nextInt();
keybd.nextLine(); // Consume the newline character

System.out.print("Enter a string: ");
String str = keybd.nextLine();
```