Introduction to programming and using Java / eclipse

Discord Link: https://discord.gg/GbVY6WTBhk

1 - The Programming Process:

Clearly define what the program’s job is

Break the process down into bite sized objectives

Use design tools to create a model of the program

Solve bite sized objectives in a manner that makes sense

Connect bite sized objectives using design model

Debug and fix any logical errors of redundancy

2 – Data Types

A ***data type*** is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.

A string, for example, is a data type that is used to classify text and an integer is a data type used to classify whole numbers

data type	definition	example
Int	integer, whole number.	3
float	floating point decimal number	12.7
double	a more precise decimal number	23.6
boolean	true or false	true
char	a single character	'a'
String	a string of characters resulting in a word / sentence	"hello"

3 – Mathematical Operators

Operation	symbol	explanation	example
Addition	+	use to add 2 numbers	num1 + num2 || num1 += num2
Subtraction	-	use to substract	num1 - num2 || num1 -= num2
Multiplication	*	use to multiply	num1 * num2 || num1 *= num2
division	/	use to divide	num1 / num2 || num1 /= num2
modulo	%	use to get remainder	num1 % num2 || num1 %= num2
increment	++	use to increment number by 1	num ++
decrement	--	use to decrement number by 1	num --

Order of operation:

from left to right, in the following order:

Operations in parenthesis

Multiplication and division

Addition and subtraction.

4 - Comparison Operators:

comparison	symbol	example	return
equality	= =	7 == 3	false
inequality	! =	7 != 3	true
less than	<	7 < 3	false
greater than	>	7 > 3	true
less than or equal to	<=	7 <= 3	false
greater than or equal to	>=	7 >= 3	true

5 - Logical Operators:

Logical Operators	symbol	explanation
AND	&&	both statements need to be true in order to return true
OR	||	either statement needs to be true in order to return true
NOT	!	statement must be false in order to return true

Examples:

Logical Operators	statement	return
AND	true statement && true statement	true
	true statement && false statement	false
	false statement && false statement	false
OR	true statement && true statement	true
	true statement && false statement	true
	false statement && false statement	false
NOT	! true statement	false
	! false statement	true

6 - Control statements:

Statements that control the flow of a program based on a conditional or value.

If statements:

Use ***if*** to specify a block of code to be executed, if condition is true.

Use ***else if*** to specify a new condition to test, if the first condition is false.

Use ***else*** to specify a block of code to be executed, if all other conditions are false.

< if / else if / else > Statements	

`if(conditional_1 == true)`
`{`
`   doSomething();`
`}`
	if conditional_1 is true, we execute doSomething()

`else if(conditional_2 == true)`
`{`
`   doSomethingElse();`
`}`
	if conditional_1 is false and conditional_2 is true, we execute doSomethingElse()

`else`
`{`
`   doNothing();`
`}`
	if both conditional_1 and conditional_2 are false, we execute doNothing()

Switch statements:

Switch Statements:

```
int variable = 0;


switch(variable)
{
      case 0:
               System.out.println("var = 0");
               break;

      case 1:
               System.out.println("var = 1");
               break;

      case 2:
               System.out.println("var = 2");
               break;

      default:
               System.out.println("var is != 0, 1, or 2");
               break;
}
```

- Generally, more efficient than nested if statements

- Test expressions based on single integer, enumerated value or String object.

- Works by passing the switch a variable to be tested.

7 - Loops:

Loops allow us to iterate a block of code repeatedly until a certain condition is met.

for loop:

```
for(initialization, condition, incrementation)
{
     instructions to be repeated.
}

e.g:

for(int i = 0; i < 10; i++)
{
     System.out.println("i = " + i);
}

output:

i = 0            // first iteration
i = 1            // second iteration
i = 2            // third iteration
i = 3            // forth iteration
i = 4            // fifth iteration
i = 5            // sixth iteration
i = 6            // seventh iteration
i = 7            // eigth iteration
i = 8            // ninth iteration
i = 9            // tenth iteration
~~i = 10~~           // no iteration, i is not < 10
```

While Loop:

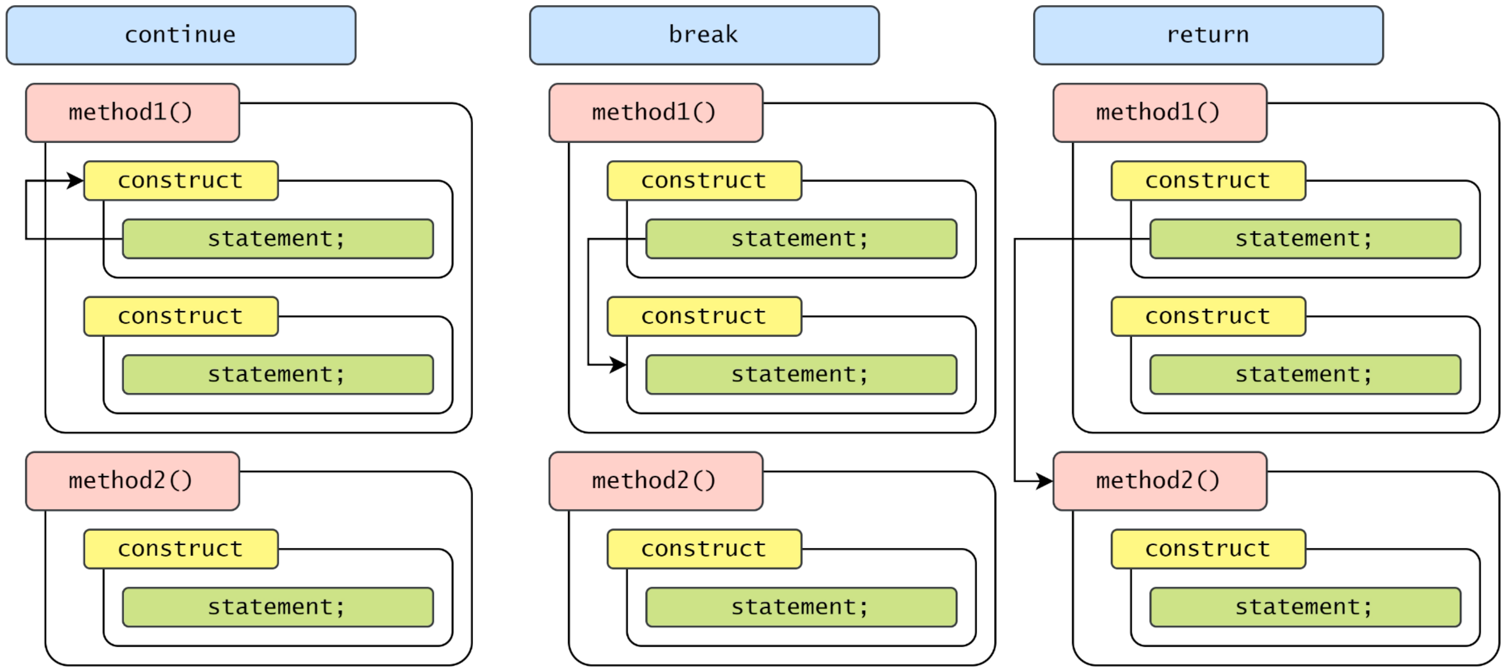
```
while(condition == true)
{
     instructions to be repeated.
}

e.g:

~~while(valid != true)~~
~~{~~
~~     instructions();~~
~~}~~                              infinite loop

boolean valid = false;    <--  initialization

while(valid != true)
{
     instructions();

     if(condition) { valid = true; }    <--  exit loop logic
}
```

8 - Break / Continue / Return:

Continue: used to move the control to the next iteration of a loop.

Break: used to exit from a loop or switch case.

Return: exit out of a function / returns value from function.



9 – Arrays:

A data structure consisting of a collection of elements identified by at least one array index or key. Used to implement other data structures such as lists and Strings.

Arrays are initialized with a fixed number of elements.

Arrays **cannot** increase / decrease in size.

For an integer containing 7 elements: int[] arr = new int[7];

arr[7] = null pointer exception, outside of array error.

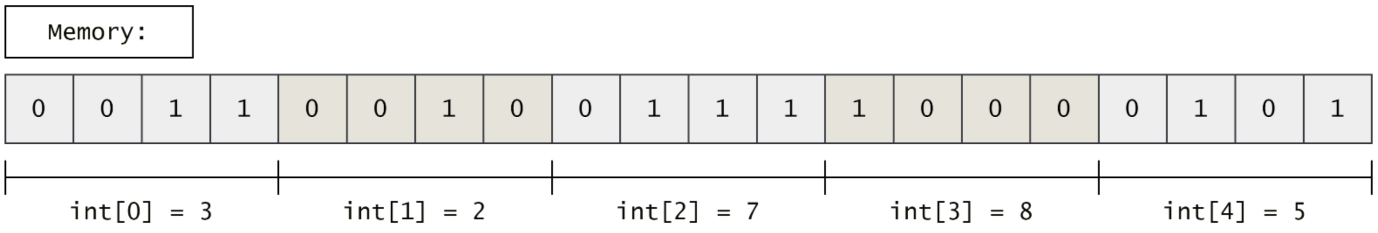
arr[-1] = null pointer exception, outside of array error.

Initializing arrays:

`String[] strCollection = new String[10];`	instantiates an array of 10 Strings, NOT initialized with any values. ie: NULL
`int[] integers = {0, 1, 2, 3, 4};`	instantiates an array of 5 int data types, initialized with values 0, 1, 2, 3, 4.

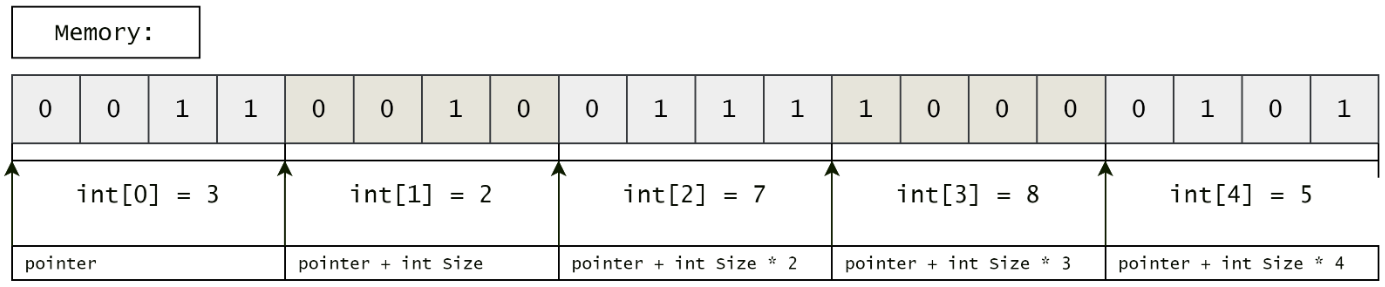
Arrays in memory:

```
int[] integers = {3, 2, 7, 8, 5};
```



Accessing array elements:

```
int[] integers = {3, 2, 7, 8, 5};
```



10 – Methods:

A method is a set of instructions inside a method block that can be called for execution using the method name and signature. A Java method can take in data or parameters and return a value.

Methods:

Main Method
< Driver >

```
public static void main(String[] args)                                  // main method
{
    System.out.println("our next line of code will call a method");     // print instruction

    mtd1();                              // calls method "mtd1" by name, and signature "()"
}
```

Method
< mtd1 >

```
public static void mtd1()                // start of mtd1 method
{
    System.out.println("in mtd1");       // print instruction
}
```

Method Parameters / signatures:

Methods are called by name and signature. A method's signature is the parameters it accepts.

Methods:

Main Method
< Driver >

```
public static void main(String[] args)                                  // main method
{
    System.out.println("our next line of code will call a method");     // print instruction

    addition();         // ERROR, need to pass params, this method does not exist

    addition(2,4);      // calls addition method
}
```

addition method

```
public void addition(num1, num2)
{
    System.out.println(num1 " + " num2 " = " + (num1 + num2));          // print instruction
}
```

<u>Method Returns:</u>

Methods that **<u>DO NOT</u>** return a value, are void methods.

However, we can write methods that return any data type, object, or data structure.

If a method has a return, then we **<u>HAVE</u>** to return something.

Methods:

void method < NO RETURN >

```
public void addition(num1, num2)
{
    instruction();              // instructions
}
```

int method < returns int >

```
public int addition(num1, num2)
{
    return (num1 + num2);       // return instruction    -    returns an integer
}
```

String method < returns String >

```
public String getName()
{
    return "Albert";            // return instruction    -    returns a String "Albert"
}
```

<u>Static vs. non-Static Methods:</u> (skip this part until we cover classes)

Static methods are methods that do not need to be instantiated as part of an object. As an example, our driver method is static.

Static methods belong to a class, NOT an object of that class.

Non-static methods belong to an object, instantiated from a class.

As an example, student1.getGrade();

We create a Student object, called student1, we then call the method getGrade() in the Student object.

The getGrade function is written in the student class but cannot be called without it being part of a student object.

Recursive Methods:

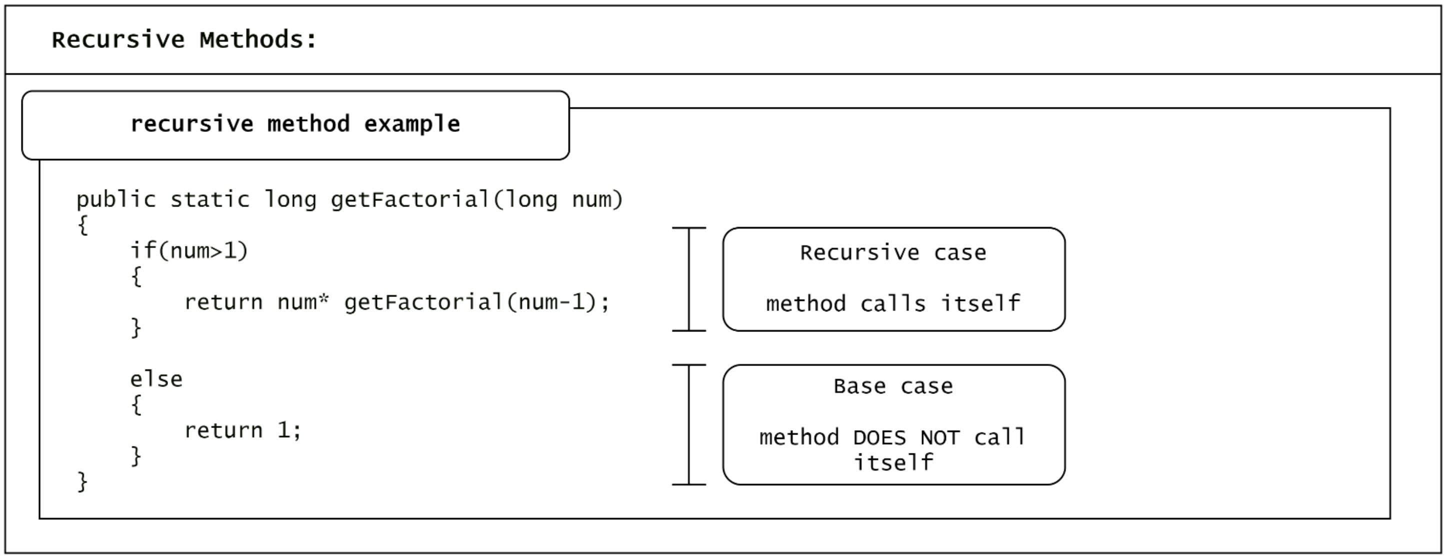
Recursive methods are methods that call themselves.

Recursive methods are a looping structure.

Recursive methods must have 2 states, a base case and a recursive case

Base cases allow the method to stop calling itself and exit the loop.

Recursive cases is when the state in which the method calls itself for recursion.



```
public static long getFactorial(long num)
{
    if(num>1)
    {
        return num* getFactorial(num-1);
    }

    else
    {
        return 1;
    }
}
```

String Manipulation:

The only way to get better at String manipulation is practice.

https://www.w3resource.com/java-exercises/string/index.php

have fun!

The Programming Paradigms:

There are many programming paradigms such as Procedural, Functional, Scripting, Logical and finally, object-Oriented.

Java is an Object-Oriented programming language; meaning, java organized data and functionality around the usage of objects.

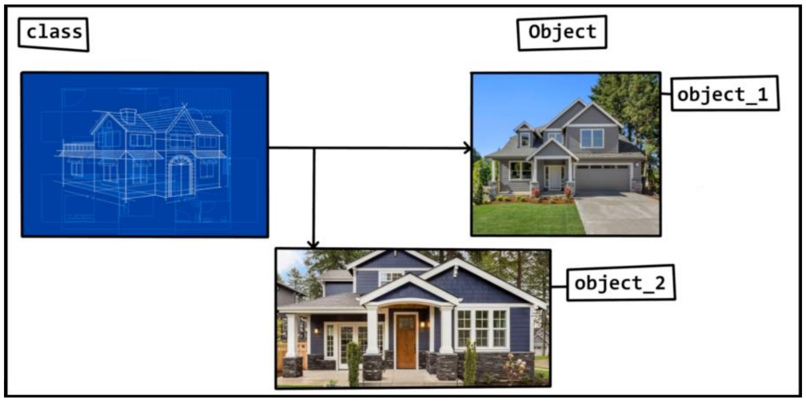
Examples of objects we have utilized so far, <Scanner> and <Random>

The 4 pillars of Object Oriented Programming (OOP):

Encapsulation:	bundling data and operations on that data into a class, used to hide and protect these data and methods. prevents unauthorized access to the data and methods of that class.
Abstraction:	abstraction is the simplification of the programming process by only revealing necessary processes and info when programming outside of a class. This allows programmers to create a simple interface in which we interact with an object and limit damage done to that class when programming outside of it.
Inheritance:	allows classes to inherit data and functionality from other classes. The inheriting class is called the child class. The class from which the child class inherits is known as the parent. This allows programmers to create multiple classes that inherit functionality rather than have to re-write the same functionality in multiple classes.
Polymorphism:	poly = many, morph = forms. Many form – ism? this allows us to have methods operate differently based on which object called the method.

Classes & Objects:

A class is a user defined structure that acts as a blueprint for objects we create. Objects are instantiated from classes and used in our code.



A house class would act as a blueprint for any house object instantiated from that class.

Anatomy of a Class:

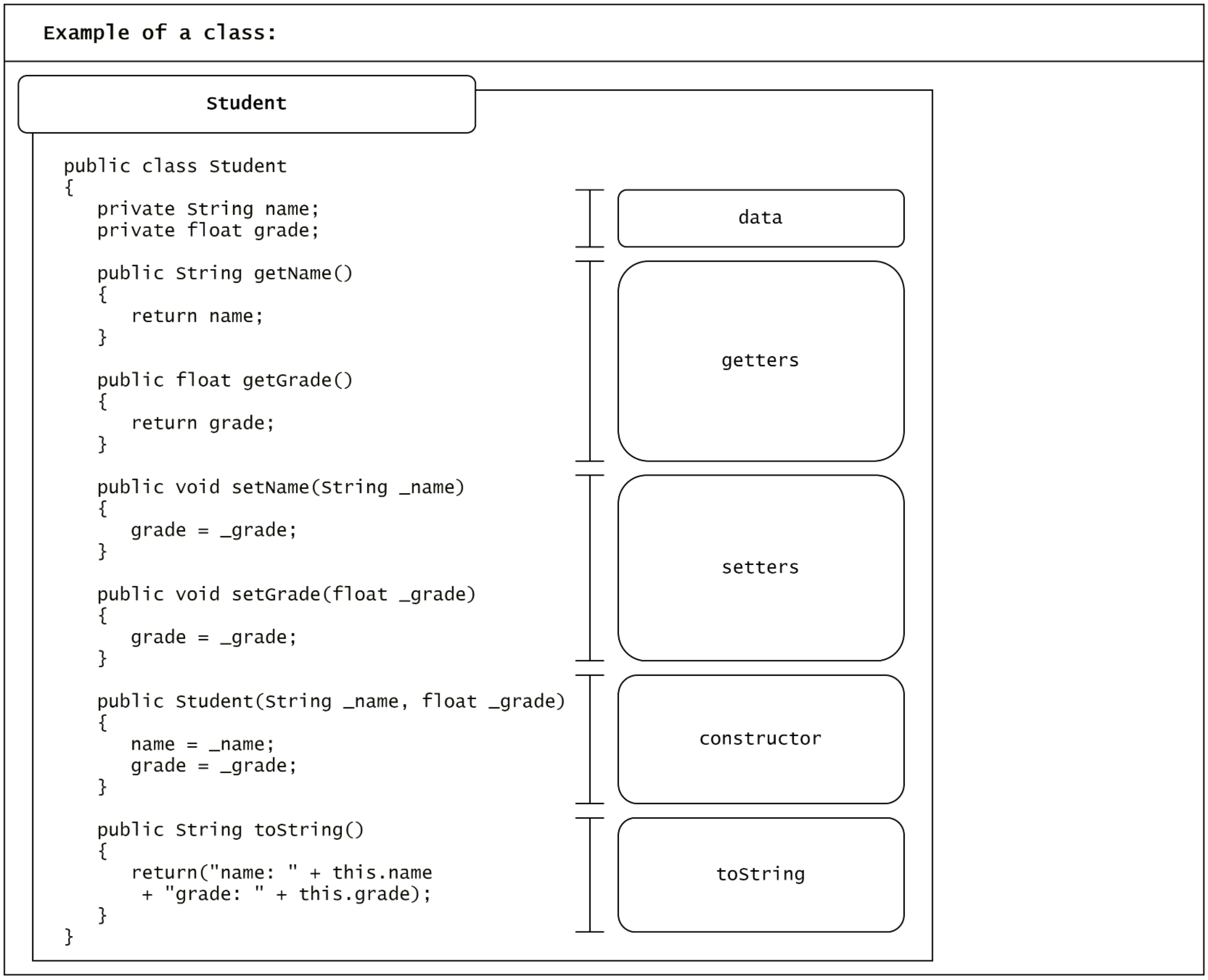
data: the data section of a class is where we instantiate variables of that class. If we create a square class in java, variables such as height and width are data of that class.

Getters: getters are functions in a class that allow us to set the variables as private, yet still have access to their values. Getters get the values of variables in a class, and pass them back to whatever function called on them.

Setters: similar to getters, setters are functions in a class that allow us to set variables as private but still have access to them. Unlike Getters however, setters DO NOT receive values, they mutate them.

Constructors: constructors are functions found in a class that allow us to create objects from that class's blueprint. it is common to create multiple constructors in a class by overloading them.

toString: the toString method is a function of a class. We can override that function by creating a toString method of our own and returning any String object of our choosing. When we pass our object in a <System.out.println> statement, wed get whatever String return we specified in our toString method, rather than the memory address of that object.



Example of a class:

Student

```
public class Student
{
   private String name;
   private float grade;

   public String getName()
   {
      return name;
   }

   public float getGrade()
   {
      return grade;
   }

   public void setName(String _name)
   {
      grade = _grade;
   }

   public void setGrade(float _grade)
   {
      grade = _grade;
   }

   public Student(String _name, float _grade)
   {
      name = _name;
      grade = _grade;
   }

   public String toString()
   {
      return("name: " + this.name
       + "grade: " + this.grade);
   }
}
```

Inheritance:

In Inheritance, the properties of the base class are acquired by the derived classes when we use the extends keyword.

e.g. public class student extends user.

user(super-class) → student(sub-class).

e.g. public class spider extends insect.
     public class brownRecluse extends spider.

insect(super-class) → spider(sub-class) → brown recluse(sub-class).

In Java, a class can inherit attributes and methods from another class. The class that inherits the properties is known as the sub-class or the child class. The class from which the properties are inherited is known as the superclass or the parent class.

Abstract Classes:

An abstract class is a class that is declared abstract. It may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An instance of an abstract class cannot be instantiated.

We can have an abstract class without any abstract methods.

We are not allowed to create objects for an abstract class.

If a class contains at least one abstract method then it has to be declared as an abstract class.

If the Child class is unable to provide implementation to all abstract methods of the Parent class, then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method.

Abstract Methods:

Sometimes, we require just method declaration in super-classes.

This can be achieved by specifying the abstract type modifier for these methods.

Abstract methods have no implementation specified in the super-class. Thus, a subclass must override them to provide method definition.

Interfaces:

An interface in Java is a blueprint of a class. A Java interface contains static constants and abstract methods

Interfaces:

An interface in Java is a blueprint of a class. A Java interface contains static constants and abstract methods.

There can be only abstract methods in the Java interface.

Interfaces are used to achieve abstraction and multiple inheritance in Java using the implements keyword.

e.g. public class spider extends insect implements arachnids.

    Insect(super class) → spider(sub class)

    Arachnids(interface) → spider(sub class)

Exception Handling:

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException...

We can write our own Exception classes by inheriting from exceptions.

Exception class

```java
public class InvalidInput extends Exception
{
   public InvalidInput(String _error)
   {
      super("Invalid Input: " + _error);
   }
}
```

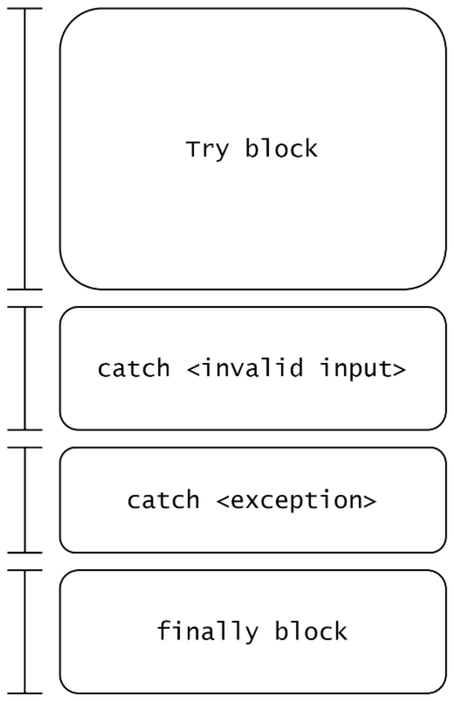
We can throw exceptions by using try, catch and finally in our program. We can also have multiple catch blocks per try block.

If we have a try block of code, then we must have a catch block of code. We do not require a finally block of code.

Try is the area in which we attempt to run some code, if the code throws an exception, we run the catch block of code. Regardless of outcomes, exception of not, the finally block gets executed.

try / catch / finally

```java
try
{
   name = sk.nextLine();

   if(name.length()<3 )
   {
       throw new InvalidInput("name not long enough");
   }
}

catch(InvalidInput e)
{
   System.out.println(e.getMessage());
}

catch(Exception e)
{
   System.out.println("Invalid Input: mistakes were made");
}

finally
{
   System.out.println("in finally block");
}
```

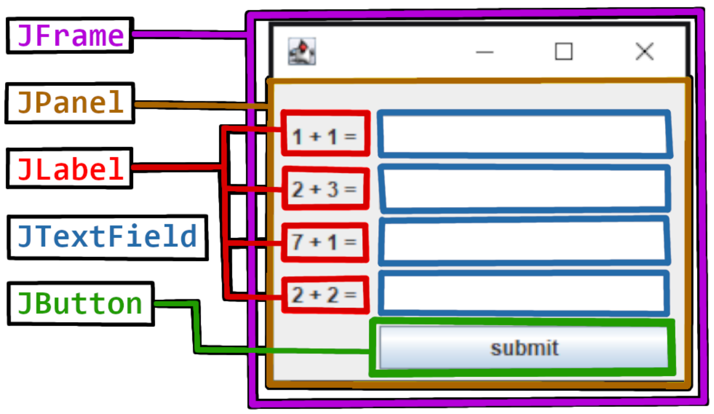


GUIs using swing in java:

Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components.

We start with a **frame** and **panel**. The frame is our window, the panel is a section, inside the frame, that holds our components.

We can use **JLabel** objects to display text.



We can use **JTextField** objects to accept user input.

We can use **JButton** objects to push an action performed action.

ArrayLists:

ArrayLists in java are similar to arrays, the difference being, we can resize an arraylist by adding and removing items from it.

Just like arrays, the JVM (java virtual machine) allocates memory in our machines to store the elements of these data structures.

However, while arrays are unmoving, array lists are.

When ArrayLists reach 75% capacity, the JVM automatically moves our arraylist to a different location in order to allocate more memory.

The elements are still stored contiguously, (one after the other), but we have the ability to resize them.

Read and Write to .txt files:

In order to find a file's directory in java, we use,
```
String path = new File("fileName").getAbsolutePath();
```

We can check if a file exists by using the exists() function
```
ourFile.createNewFile();
```

We can write data to a file by using the fileWritter object.
```
FileWriter writer = new FileWriter(ourTxtFilePath);
writer.write(allNames);
writer.close();
```

Finally, we can read data from .txt files by using our Scanner object.

```
Scanner sk = new Scanner(ourFile);
```