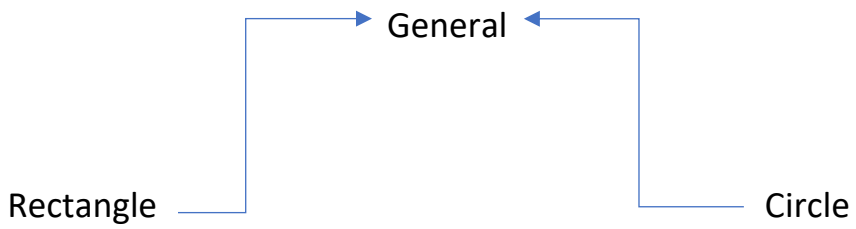
Answer 1

Suppose, I want to work with some real-life geometric object shaped like circle, rectangle, square etc. They have some common features/attributes like color, date of manufacturing etc. And they have some individual attributes like height, width, radius depending on their shape. They might have individual methods and they have some common methods like methods to calculate area, calculate perimeter etc. In future I may have to introduce new objects with new geometric shapes.

With the help of OOP, I can design the classes with their attributes, so that new objects can be created of that class. Class is a general structure of the same like objects.

We will have a class named **General (**assume**)** which includes the common features and methods. The other class like **Circle** and **Rectangle** inherits the attributes features of the parent class plus includes their own attributes. This is called **Inheritance.**



The area/perimeter calculation formula for different shapes are different. So, the implementation of that methods works with same name but with different implementation. This scenario is called **Polymorphism** in OOP.

Circle

```
getArea() {
  return PI*radius*radius;
}
```

Rectangle

```
getArea() {
  return width*height;
}
```

We want to restrict outer direct access to some of our object component. So, we can modify access levels with private, public and protected. This is called **Encapsulation.** For example, we want to fix all the circular objects radius is 8 for future manufacturing. We want to keep this data read only. So we can declare the radius as **Private** and write a getter method to read it from outside world.

```
private int radius=8;
public int getRadius()
{
```

```
        return radius;
    }
```

Run time Polymorphism

Polymorphism means having different forms. In OOP, method overriding is called run time polymorphism. In method overriding, a subclass overrides a method with the same signature as that of in its superclass. Unlike compile time polymorphism, it resolves in run time that means the method executes based on which object will execute it.

```
Class A{
     Void print(){
          Print("Lukaku scores");
       }
 Class B{
    Void print(){
     Print("Lukaku scores 2 goals");
}
Public Class Main{
       Public static void main(String[] args)
```

```
{
    A a = new A();
    A b = new B();
   a.print();              // This will print “Lukaku scores”
   b.print();              // This will print “Lukaku scores two goals”
}
}
```

Answer 4

```
int traverse(Node* node){
    if (node == null) return 0;          //base case
    count++;
    count += traverse (node->left)       //recursive call
    count += traverse (node->right)      //recursive call
    return count;
}
```

There are 2 semicolons missing. By the way, it will over count the number of nodes**.** Simply traversing the left and right subtree and managing a counter value will do. The unnecessary parts are being highlighted.

Another approach: As the tree is heavily left skewed and million of nodes are there, **we can reduce recursive call only calling the left subtree**. Some of the nodes might have a right child but no right subtree as it is heavily left skewed. So, we can keep a counter of right child.

```
Int traverse (Node* node) {
    If(node==null) return 0;
    count++;
    If(node->right!=null)
    right_child_count++;
    traverse(node->left);
    return count+right_child_count;
}
```

Answer 3

Suppose the work is continuing in a 4-core processor. So, we can do a parallel processing to speed up the matrix operation. Here we assume two same dimensional (4x4) matrix and process the calculation parallelly to speed up the process.

A[0][0]*B[0][0], A[0][0]*B[0][1], A[0][0]*B[0][2],……,A[0][0]*B[3][3]	Processor 1
A[1][1]*B[0][0], A[1][1]*B[0][1], A[1][1]*B[0][2],……,A[1][1]*B[3][3]	Processor 2
A[2][2]*B[0][0], A[2][2]*B[0][1], A[2][2]*B[0][2],……,A[2][2]*B[3][3]	Processor 3
A[3][3]*B[0][0], A[3][3]*B[0][1], A[3][3]*B[0][2],……,A[3][3]*B[3][3]	Processor 4

N.B: This is an idea. The technical terms might be different than this.

Answer 2

Stack Memory

Stack memory is the space where local variables, arguments get space. The backend mechanism is like allowing the system memory to be used as temporarily data storage. It behaves like first in last out buffer. Stack memory allocation and de-allocation happens automatically as soon as the relevant method finishes its execution.

Faster, sequential spaced and less memory space compared to Heap memory.

Heap Memory

Larger space available to allocate/de-allocate manually. It is the dynamic memory that can grow and shrink with our need. Comparing to stack memory, it is slow and random spaced. It needs to manage carefully to prevent data leakage and fragmentation.

Considering need of a large array (100MB), it might make more sense to store in a heap memory rather than stack memory. But allocating and de-allocating creates complexity and if we need those functions to be performed faster using this allocated memory in some scenarios, we can do allocation in stack memory.