RTTI -> Type Information

typeid(Rectangle)-> Operator -> type_info class

dynamic_cast
static_cast
reinterpret_cast
const_cast

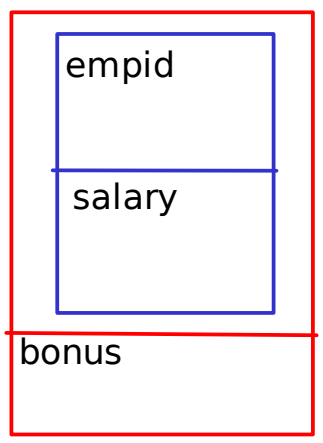
reinterpret_cast p

```
Employee e;
e.getName();
e.getEmpid();
e.getSalary();
```

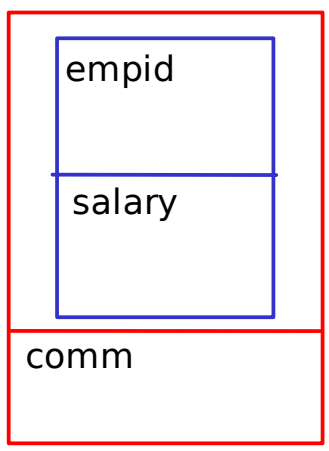
name() -> inspector or getter

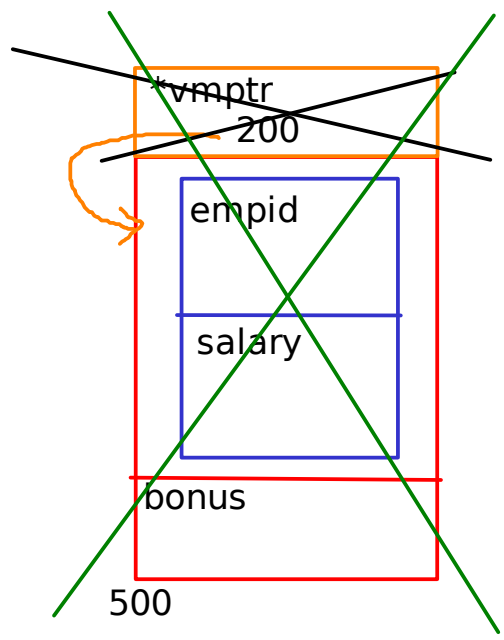
```
Rectenagle robj;
typeid(robj).name()
```

manager




salesman




```
accept(){
employee::accept()
bonus
}
```

```
accept(){
employee::accept()
comm
}
```

```
Manager obj;

Employee *eptr = new Manager();
delete eptr;
```

Interface

```
// abstract class
class Shape{
double area;

double getArea(){
return area;
}

virtual void acceptData()=0;
virtual void calculateArea()=0;
}
```

```
// Interface
class Shape{
virtual void acceptData()=0;
virtual void calculateArea()=0;
}
```

Exception Handling

try
throw
catch

Why?
-> To seperate Business Logic From Error handling Logic

```
// Errorhandling
if(resourcce found){

}
else{

}
```

```
// Business Logic
if(rollno == student.getRollno){
{
}
else{

}
```

```
int main(){

int a,b;
a = 10;
b = 20;
int res = a + b;
cout<<res;
}
```

```
int main(){

int a = 10;
int b = 20;
cout<<a +b;
}
```

```
int main(){

int a = 10;
int b = 20;
add(a,b);
}
```

```
void fun(){
// Generate Exception
keyword -> throw type
}
```

Custom Exception class

```
input(){


}
```

```
try{
      division();
}
catch(){
try{
input();
}catch
}
```

```
int main(){

try{
fun(); -> throw
}catch(type ){

}

}
```

```
try{
      try{
            division();
      }
      catch()
      {
            if(deno<0)
                  throw 1;
      }
}
catch(int e){
cout<<endl;
}
```

Exception Specification List

Template-> Generic code in cpp

STL

```
templete<class T>
class Array{
int size;
int index;
T *ptr
}
```

Array<?> a1(5);

function template

```
template<typename T>
void swap(T &n1, T &n2){
T temp = n1;
n1=n2;
n2=temp;
}

int main(){
Employee n1 = 5.12;
double n2 = 10.11;
swap(n1,n2);

cout<<n1<<n2;

}
```