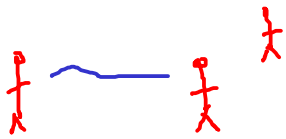
Coding convesion:
name of class should start with capital letter
PartTimeEmployee

parttime_employee

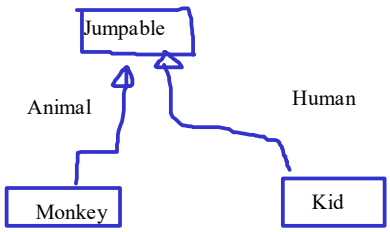
data
payPerHr

When to use what?
abs class vs interface



Monkey
jump()
stayOnTree()
bite()

Kid
jump()
school()
...



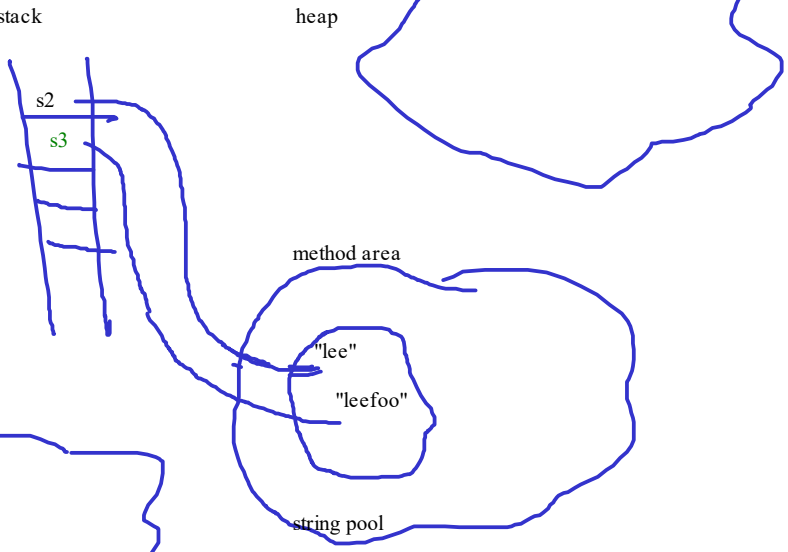
Strings are immutable (once it created there state can not be change)

How String stored in java?

String s1=new String("lee");
or
String s2="lee";

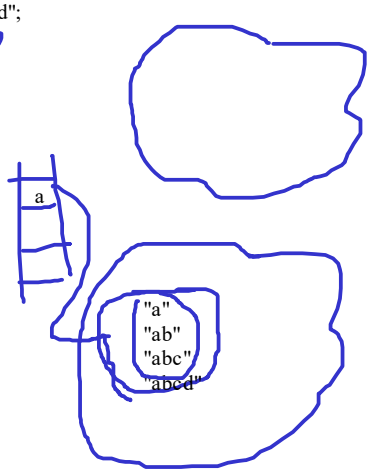
String s3="lee"

s3=s3+"foo"

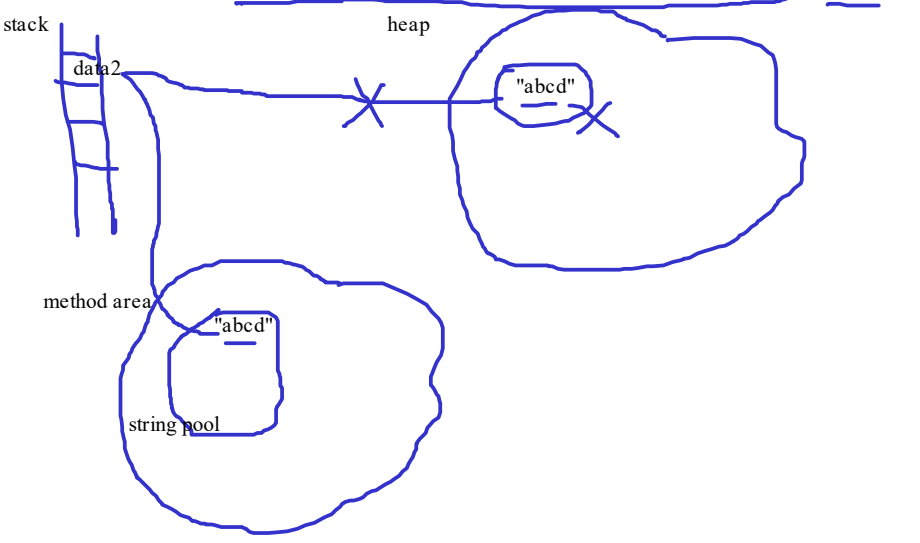


Bad prog practice

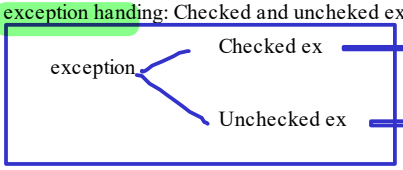
String a="a+"b"+"c"+"d";



String data2=new StringBuilder().append("a").append("b").append("c").append("d").toString();



exception handing: Checked and uncheked ex? GPP(good programming practice)

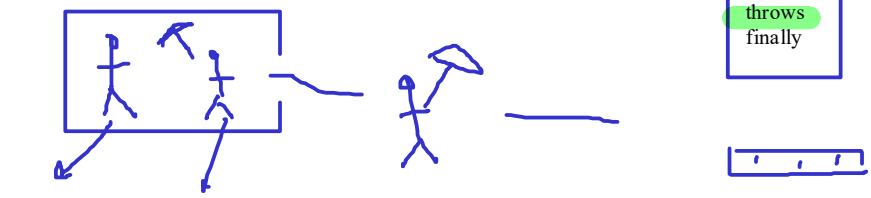


Compile time ex

run time ex

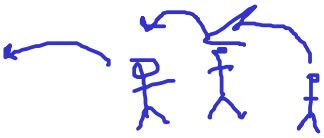
All Ex happend at run time?

try
catch
throw
throws
finally



"God give me the problem
but give me strenght to handle that"

"robust"



Examples: divide method


```
try {
	int x, y;
	Scanner scanner=new Scanner(System.in);

	System.out.println("PE 2 nos");
	x=scanner.nextInt();
	y=scanner.nextInt();

	int z=x/y;

	System.out.println("Z:"+z);

	System.out.println("end");
}
catch(ArithmeticException e) {
	System.out.println("dont do divide by zero");
}
catch(InputMismatchException e) {
	System.out.println("input must be ints");
}
```

```
Scanner scanner = null;
try {
	int x, y;
	scanner = new Scanner(System.in);

	System.out.println("PE 2 nos");
	x = scanner.nextInt();
	y = scanner.nextInt();

	int z = x / y;

	System.out.println("Z:" + z);

	System.out.println("end");
	scanner.close();

} catch (ArithmeticException e) {
	System.out.println("dont do divide by zero");
}
catch (InputMismatchException e) {
	System.out.println("input must be ints");
}
```

DRY

User define exception:
throw vs throws
try
catch
throw
throws
finally

Need of user define ex?
bankapp

Account

withdraw
deposit
account creation process

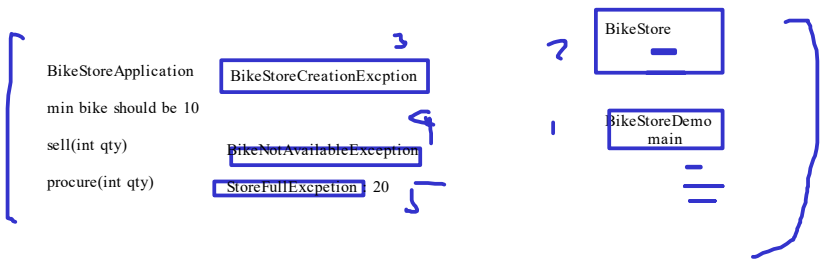
SA

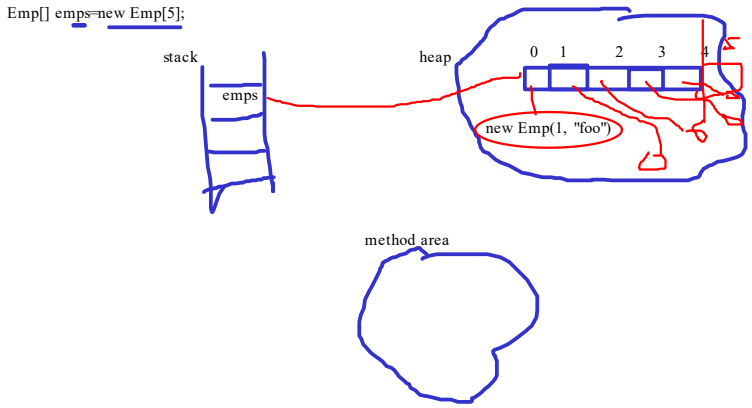
1000

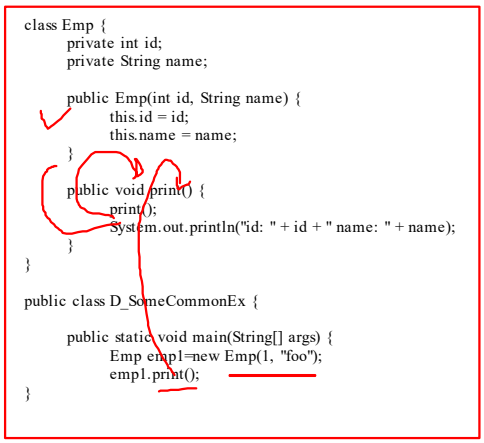
NotSufficientFundException
OverFundEcxception
AccountCreationException

```
public Account(int id, String name, double balance) throws AccountCreationException{
    if(balance<1000) {
        throw new AccountCreationException("account can not be created min bal must be 1000 usd");
    }
    this.id = id;
    this.name = name;
    this.balance = balance;
}
```

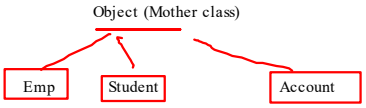
```
class AccountCreationException extends Exception{
	private static final long serialVersionUID = 1825616456008488982L;

	public AccountCreationException(String message) {
		super(message);
	}
}

//NotSufficientFundException
class NotSufficientFundException extends Exception{
	private static final long serialVersionUID = 1825616456008488982L;

	public NotSufficientFundException(String message) {
		super(message);
	}
}

//OverFundEcxception
class OverFundEcxception extends RuntimeException{
	private static final long serialVersionUID = 1825616456008488982L;

	public OverFundEcxception(String message) {
		super(message);
	}
}

class Account{
	private int id;
	private String name;
	private double balance;

	public Account(int id, String name, double balance) throws AccountCreationException{
		if(balance<1000) {
			throw new AccountCreationException("account can not be created min bal must be 1000 usd");
		}
		this.id = id;
		this.name = name;
		this.balance = balance;
	}

	public void withdraw(double amount)throws NotSufficientFundException {
		double temp=balance-amount;
		if(temp<1000) {
			throw new NotSufficientFundException("transaction is not possible min bal should be 1000 usd");
		}
		else
			balance=temp;
	}
	//1L
	public void depsit(double amount)throws OverFundEcxception {
		double temp=balance+amount;
		if(temp>=10_00_00) {
			throw new OverFundEcxception("transaction is not possible max bal should be less then 10_00_00 usd");
		}
		else
			balance=temp;
	}
	public void print() {
		System.out.println("id: "+ id +" name: "+ name +" balance: "+balance);
	}
}
public class A_UserDefineEx {

	public static void main(String[] args) {
		Account account;
		try {
			account = new Account(1, "pol", 900);
			account.depsit(5000);
			account.withdraw(94500);

			account.print();
		} catch (AccountCreationException e) {
			System.out.println(e.getMessage());
		}catch(OverFundEcxception e) {
			System.out.println(e.getMessage());
		} catch (NotSufficientFundException e) {
			System.out.println(e.getMessage());
		}


		System.out.println("code end");

	}

}
```

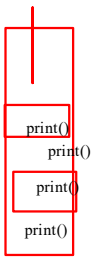






```
class Emp {
        private int id;
        private String name;

        public Emp(int id, String name) {
                this.id = id;
                this.name = name;
        }
        public void print() {
                print();
                System.out.println("id: " + id + " name: " + name);
        }
}

public class D_SomeCommonEx {

        public static void main(String[] args) {
                Emp emp1=new Emp(1, "foo");
                emp1.print();
        }
```





why Object class?

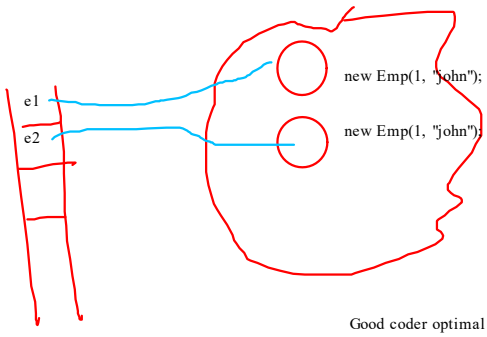
```
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws java.lang.CloneNotSupportedException;
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final void wait() throws java.lang.InterruptedException;
  public final void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
}
```

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

com.session2.ob_class.Emp@50040f0c

```
Emp e1=new Emp(1, "john");
Emp e2=new Emp(1, "john");

//compare there content
if(e1==e2) {
        System.out.println("both are same");
}else
        System.out.println("not same");
```

it check address of object

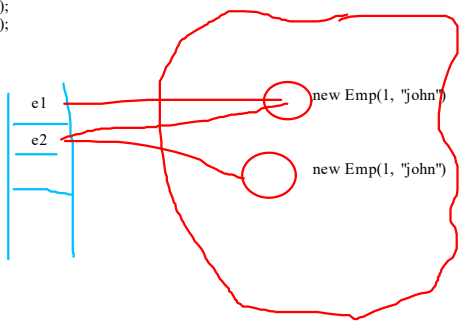


Good coder optimal code

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

```
Emp e1=new Emp(1, "john");
Emp e2=new Emp(1, "john");
```

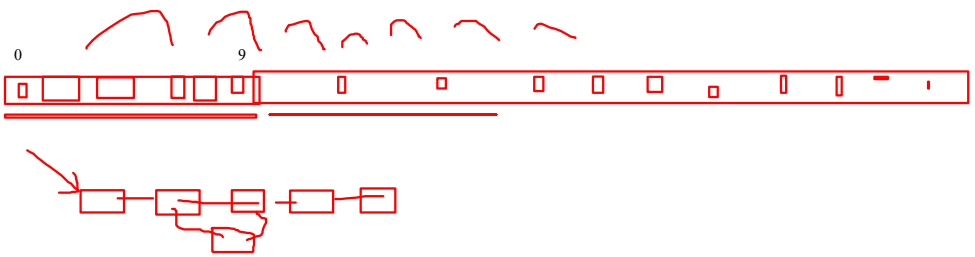
```
@Override
        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (obj == null)
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                Emp other = (Emp) obj;
                return id == other.id && Objects.equals(name, other.name);
        }
```




```
if(e1.equals(e2)) {
        System.out.println("both are same");
}else
        System.out.println("not same");
```

```
public static void main(String[] args) {
        List<Integer> list1=new ArrayList<>();
        List<Integer> list2=new LinkedList<>();
        doTiming(list1);

}
//time taken: 2111 ms for ArrayList
//time taken: 11 ms for LinkedList
private static void doTiming(List<Integer> list) {
        for(int i=0;i<1E5; i++) {
                list.add(1);
        }
        long start=System.currentTimeMillis();

        for(int i=0;i<1E5; i++) {
                list.add(0,i);
        }

        long end=System.currentTimeMillis();
        System.out.println("time taken: "+(end-start)+" ms");
}
```

100000



Set: set dont allow duplicate data
HashSet
LinkedHashSet
TreeSet