String
- String name = "sunbeam";
- String name2 = "sunbeam";
- String name2 = new String("sunbeam");

String s2,s3,s4

StringBuffer, StringBuilder

```
String s1 = "sunbeam";
s1 = s1+"Infotech";
```

Enum
- to store constant values

```
enum Aop{
EXIT(){
}, ADD(){
}
}
```

Aop.EXIT

Aop.ADD

```
Aop.values()[choice];
case ADD:
```

```
class Aop extends Enum{

}
```

```
class Constants{

}
```

Generics

Generic code

```
int[] arr = new int[5];
double[] arr = new double[5];
Employee[] arr = new Employee[5];
```

```
Employee LinearSearch(Employee arr[]){
// retrun Employee
}

SelectionSort(Employee arr[]){
//
}

Product LinearSearch(Product arr[]){
// return Product
}

SelectionSort(Product arr[]){
//
}
```

```
Car[]arr;
Point[]arr;
Product[]arr;
```

```
//DS
class Array{

T arr[]


}
```

```
Employee linearSearch(Employee [] arr, Employee key){
    for(Employee element: arr)
        if(element.equals(key))
            return element;
}
```

```
template<typename T>
T linearSearch(T [] arr, T key){
    for(T element: arr)
        if(element.equals(key))
            return element;
}
```

Before java 5

```
class Box{
    Object data;

    void setData(Object data){
    this.data = data;
    }

    Object getData(){
        return data;
    }
}
```

```
class Mobile{

}
```

```
Box b1 = new Box();
b1.setData("Mobile");
String data = (String)b1.getData(); // downcasting

Box b2 = new Box();
b2.setData(new Mobile());
Mobile m = (Mobile)b2.getData();

Box b3 = new Box();
b3.setData(new Painting());
Tv t1 =(Tv) b3.getData(); // classcastexception
```

```
// From java 5

class Box<T>{
	T data;

	void setData(T data){
	this.data = data;
	}

	T getData(){
		return data;
	}
}
```

```
	Box<Mobile> b1 = new Box<Mobile>();
	b1.set(new Mobile());
	Moile m = b1.getData(); // ??

	Box<Tv> b2 = new Box<Tv>();
	b2.setData(new Painting()); // compiler error
	Tv t = b2.getData();
```

Generics
- Generic class
- Generic Methods
- Generic Interfaces

Type Safety

```
void printArray(Object []arr){
	for(Object element: arr)
		sysout(element);
}
```

```
	<T>void printArray(T []arr){
		for(T element: arr)
			sysout(element);
	}
```

Type parameters
1. Bounded
	- are used for class types
2. unBounded
	- are used for class references

```
template<typename X, typename Y >

		class Box<T>{
			T obj;
		}
```

```
		<T>void method1(T ref){
		}

		method1(new Employee());

		method1(10);

		methid1(new Date());
```

```
		void method2(Box<? extends Number> ref){

		}

		method2(new Box<Integer>());

		method2(new Box<Employee>());

		method2(new Box<Date>());
```

Limitations of Generics
1. Cannot use primitive types for generics
2. T obj = new T(); // NOT OK
3. T[] obj = new T[5]; // NOT OK
4. throw new T(); // NOT OK
5. catch(T obj){} // NOT OK
6. private static T obj; // NOT OK
7. (obj instanceof T) // NOT OK

```
	class Number{

	}

	class Integer extends Number{

	}

	Number n = new Integer();
```

```
Class Box{
Number ref;
}

class Box extends Box{
Integer ref;
}
```

Box<Number>

Box<Integer>

Comparable
Comparator

```
Box<T extends Exception>{

}

void method(Box<? super Run..> b){

}
```

```
void sort(Object [] arr){
for( int i = ; i<arr.length; i++)
{
for(int j = i+1; j<arr.length; j++){
// condition
Comparable obj1 = arr[i];
Object obj2 = arr[j];
if(obj1.compareTo())
{
//swap
}
}
}

}
```

obj>obj2