GENERICS & COLLECTION

Java Tutorial #2

신용호

컴퓨터과학과 12

목차

들어가기

Introduction

Collection과 Generics 이해하기

자료구조 잘 알죠?

- 여기서 기본적인 자료구조에 대해서 설명하지는 않겠습니다만
  적절한 자료구조가 효율적인 프로그램을 만든다는 것은 너무도 자명합니다.

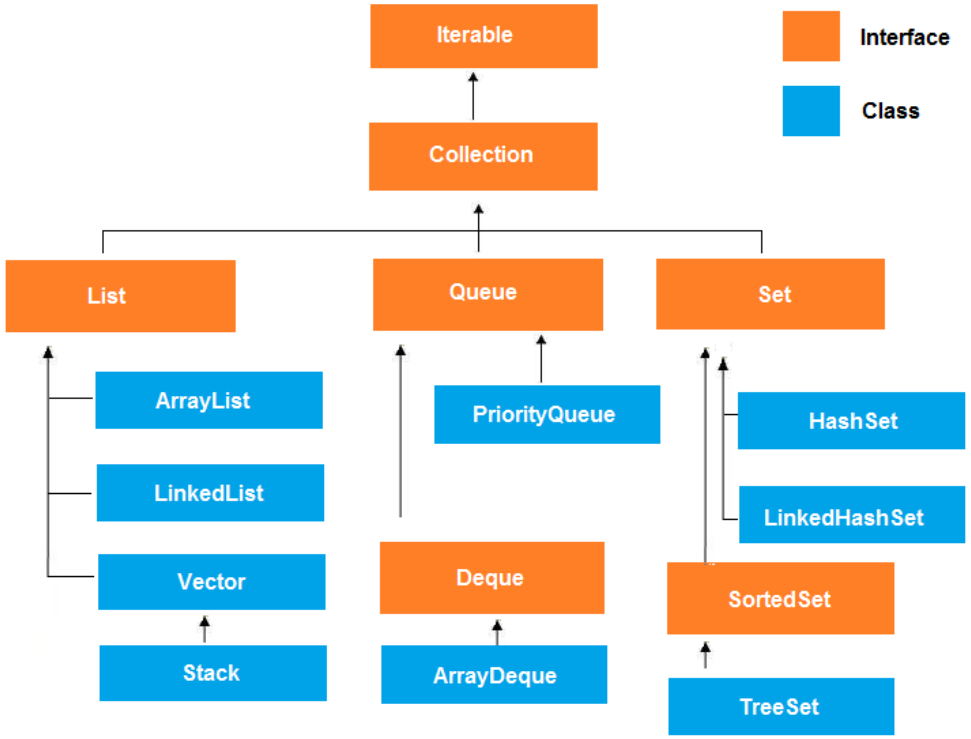
- Java와 같은 객체지향언어에서는 객체를 효율적으로 운용할 필요가 있죠.

- C에서는 구조체(struct)를 직접 가지고 놀았을 테고
  C++에서는 STL이라는 아름다운 라이브러리를 선물 받았지요.
  그러면 Java에서는?

- Java에는 Collection이 있답니다.

Collection

- 오른쪽 그림이 Java에서 제공하는 Collection을 나열한 것들입니다.
- 우리가 무엇을 더 작성해야하는 것이 아니냐고요?
  걱정 마세요. 이미 다 완성되어 있으니까요. 우리는 적절하게 사용하기만 하면 된답니다!
- 그런데 그 안에 어떤 것을 담을 줄 알고 이렇게 만들었을까요?



몰라요.

- 어떤 타입이 자료구조를 이루게 될지 미리 알 수 없지요.
- 그렇다고 모든 경우에 대해서 만들자니 너무 많기도 하고요.
- 프로그래머가 따로 구현한 객체들도 담아야 하는데 그건 어떻게 담나요?

Generics

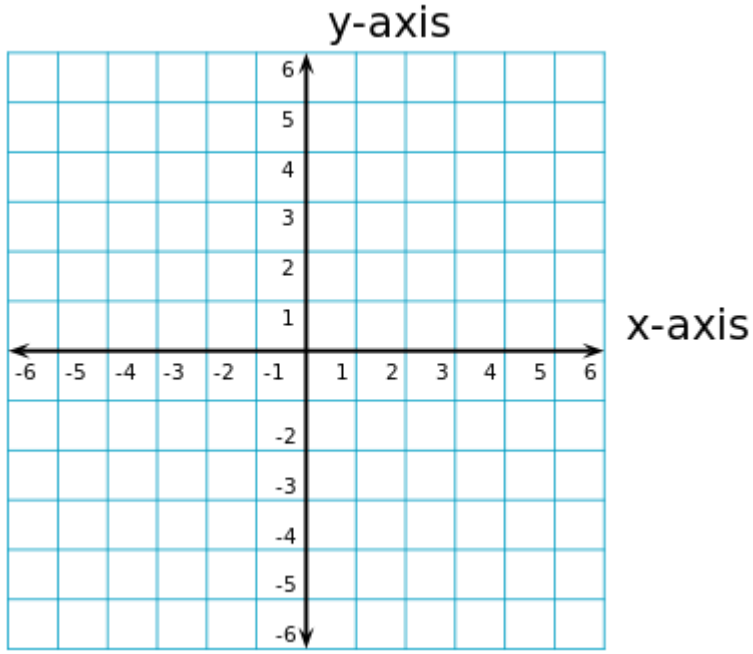
- 그냥 모른 채로 두세요
- 대신 자료 구조 안에는 모두 똑같은 타입의 객체로만 이루어져 있겠죠.

- C++에서의 Template처럼,
  Java에는 Generics라고 합니다.
- 메소드에 선언해서 해당 메소드에만 적용할 수 있고요,
  클래스에 선언해서 클래스 전체에 적용할 수도 있어요!

잠깐!

- Collection은 Generics를 사용하기 때문에
  반드시 객체만 받을 수 있어요.
- int, double과 같은 primitive type은 사용할 수 없답니다.
  대신 Integer, Double과 같은 Wrapper 클래스를 사용하면 됩니다.

들어가기 앞서



- 우리를 도와줄 객체를 하나 만들려고 해요.
  바로 정점(Point) 객체랍니다.

- 정점 객체는 double x와 double y를 필드로 가져요.

- 한번 직접 만들어 볼까요?
  Getter와 Setter도 있어야 겠고, 적절한 생성자도 만들어줘야 하겠네요.
- 자동 완성 기능을 활용하면 훨씬 편리하답니다.

이렇게 만들었나요?

```
public class Point {
	private double x;
	private double y;

	public Point (double x, double y) {
		this.x = x;
		this.y = y;
	}


	public double getX() {
		return x;
	}

	public void setX(double x) {
		this.x = x;
	}

	public double getY() {
		return y;
	}

	public void setY(double y) {
		this.y = y;
	}

}
```

main함수가 필요해요.

- 프로그램을 돌리려면 main함수가 필요하죠?
  각자 main함수를 알아서 만들어 보세요.

- 일단 Java는 모두 객체로만 구성이 되어야 하니까,
  새로운 클래스를 생성해야 겠죠?

- 클래스를 만들 때 Which method stubs would you like to create?에서
  첫 번째를 클릭하면 main함수를 자동으로 생성해 준답니다.

정점을 정의하죠!

- 이제 main함수 안에 정점을 정의하죠.
- 저는 오른쪽 코드처럼 정의하였답니다.
  꼭 저와 같이 하실 필요는 없어요.
- p3과 p5처럼 하나는 꼭 동일한 정점을
  만들어 주세요!

```
Point p1 = new Point(0f, 0f);
Point p2 = new Point(2f, 5f);
Point p3 = new Point(4f, 4f);
Point p4 = new Point(7f, 9f);
Point p5 = new Point(4f, 4f);
```

LIST<T>

첫 번째 자료구조

여러가지 반복문

Object 객체와 toString 함수

이제 자료구조를 만들어 봅시다.

- 아래의 코드를 입력해 봅시다.

```
List<Point> listOfPoints = new ArrayList<Point>();
```

- 이런 빨간 줄이 생겼어요! 왜 그런 것일까요?
  객체를 사용하려면 그 객체가 들어 있는 라이브러리가 필요하기 때문이지요.

- 그런데 직접 넣으려고요? 어디에 있는 줄 아시고….

우리 똑똑한 Eclipse~

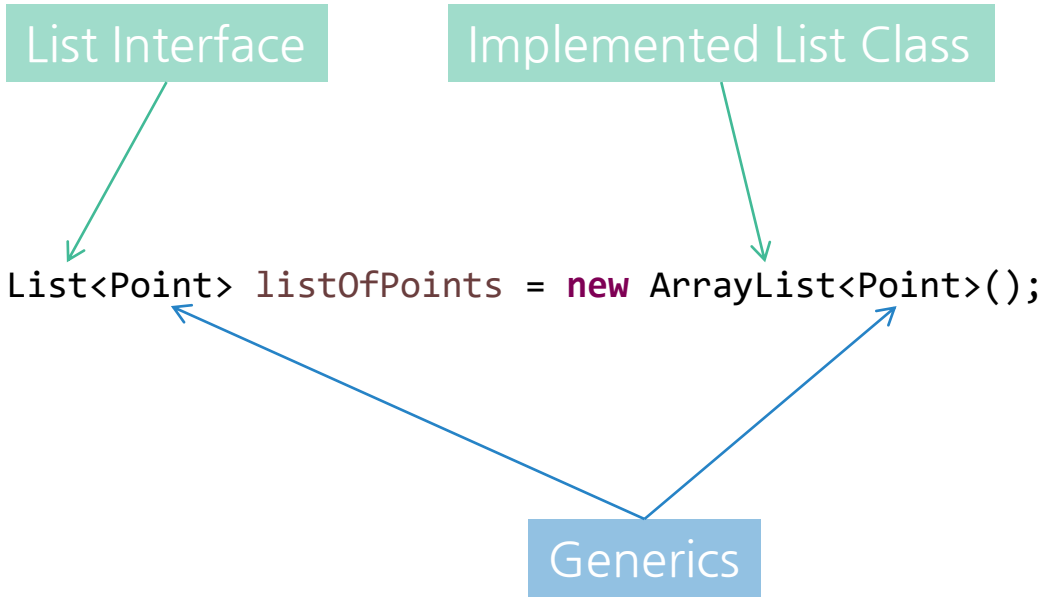
- 빨간 줄에 마우스를 갖다 대면
  창이 하나 생겨요.

- 여러 가지 항목 중에서
  Import ‘List’ (java.util) 을 클릭!
  절대 java.awt가 아니에요!

- 그러면 오른쪽과 같이 자동으로
  import된 것을 확인할 수 있어요.

```
import java.util.ArrayList;
import java.util.List;
```

좀 더 뜯어볼까요?



- List는 인터페이스에요.
  List 자료구조에 필요한 체크리스트를 담고 있지요.
- ArrayList는 Java가 기본적으로 제공해주는 List 클래스랍니다.
  이외에도 LinkedList 등이 있어요.
- 필요하다면 직접 List를 구현하는 자신만의 List 객체를 만들 수도 있겠죠?
- <…>가 Generics를 의미해요.
  여기서는 우리의 객체 Point가 들어갔네요.
- 그러니 listOfPoints가 Point로 이루어진 List 객체라는 것을 알 수 있답니다!

이제 넣어 봐야지?

```
listOfPoints.add(p1);
listOfPoints.add(p2);
listOfPoints.add(p3);
listOfPoints.add(p4);
listOfPoints.add(p5);
```

- add함수가 List에 객체를 넣을 때 쓰는 함수입니다.
- List 인터페이스에 명시가 되어있어요.
- 우리의 예제에서는 ArrayList가 구현한 add를 호출했겠지요?
  하지만 우리는 ArrayList가 어떻게 동작하는 지 알 필요는 없지요.
- 이게 객체지향의 묘미랍니다.

잘 들어갔나?

- 잘 들어갔는지 확인하기 위해서 출력을 해봅시다.

- 일단 List의 정보를 앞에서부터 하나씩 읽어서 이를 출력해야 겠지요?

- 아마 오른쪽과 같은 코드를 생각할 거에요.

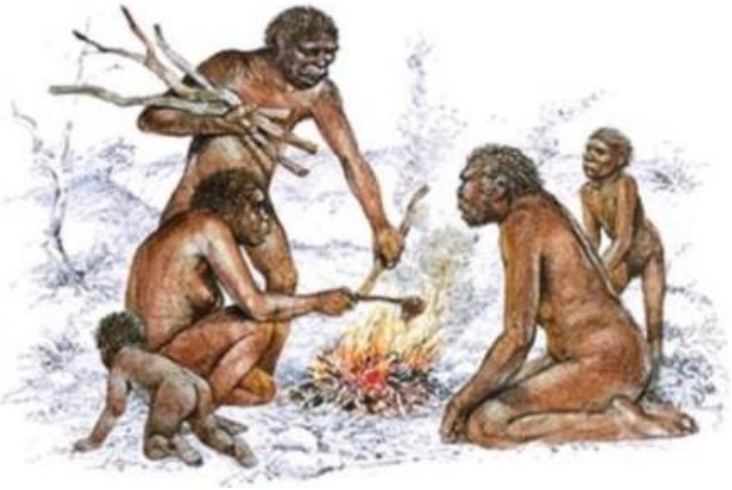
```
for(int i = 0; i < list.size(); i ++) {
     System.out.print(list.get(i)+" ");
}
```

원시인도 아니고…

- 하지만 저런 반복문은 너무 원시적(?)이지 않나요?
  절대 저 방식이 나쁘다는 것은 아닌데
  다른, 조금 더 깔끔한 방식이 있지 않을까요?

- Java에서는 여러가지 반복문 방식을 제공해 줍니다.

- 속도는 느려질 수도 있지만, 보기에 더 깔끔하고,
  때에 따라서는 훨씬 직관적으로 이해할 수 있지요.



잠깐 그 전에!

- List에 담긴 내용을 확인하는 새로운 객체를 만들도록 할게요.
- 그저 편의를 위해서라기 보다는 재사용성을 높이는 작업이라고 생각합니다.
- 이름이 반드시 똑같을 필요는 없어요. 그저 예시일 뿐이랍니다.

클래스 단위 Generics입니다.
이 클래스 안에서 T는
이 클래스 생성 시에 정의된 객체를 의미합니다!

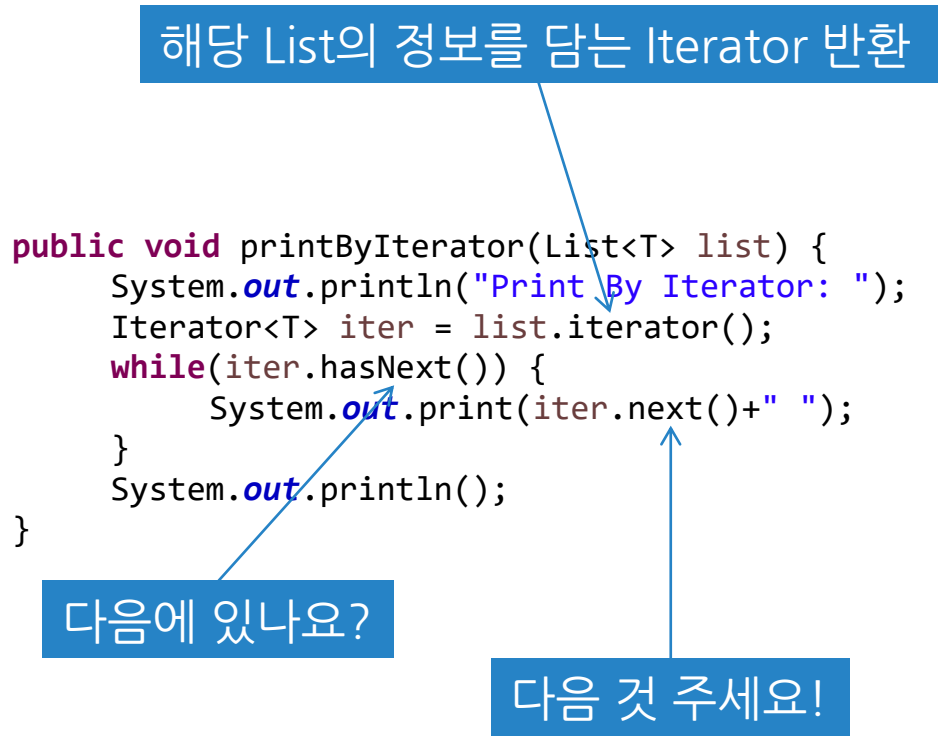
```
public class ListPrinter<T> {

	public void printByIndexedFor(List<T> list) {
		System.out.println("Print By Indexed For: ");
		for(int i = 0; i < list.size(); i ++) {
			System.out.print(list.get(i)+" ");
		}
		System.out.println();
	}

}
```

```
ListPrinter<Point> listPrinter = new ListPrinter<Point>();
listPrinter.printByIndexedFor(listOfPoints);
```

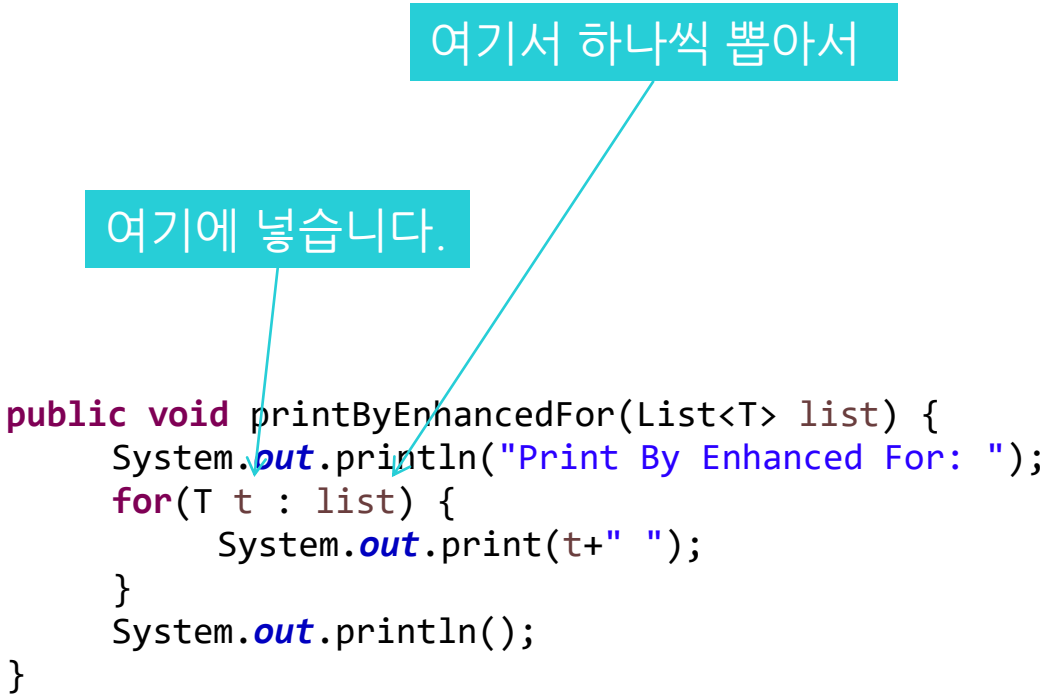
Iterator? 그게 뭐야?



- 첫 번째 방식은 Iterator를 활용하는 방법입니다.
- Iterator는 List 객체를 반복시켜주는 객체로, List의 객체를 처음부터 하나씩 받아서 차례로 참조합니다.
- hasNext 함수를 통해서 아직 남아 있는지를 파악하고
next 함수를 통해서 다음 객체를 참조합니다.

이런 for는 처음이야~

- 두 번째 방식은 enhanced for라고 불리는 방식입니다.
- 말 그대로 콜론 뒤의 List 객체에서 앞의 변수에 하나씩 뽑아 주는 것이랍니다.
- 이 방식은 꼭 List 뿐만 아니라, 기본 타입 배열 (int[], double[]) 등에서도 쓸 수 있어요.




```
public void printByEnhancedFor(List<T> list) {
      System.out.println("Print By Enhanced For: ");
      for(T t : list) {
            System.out.print(t+" ");
      }
      System.out.println();
}
```

실행을 했는데…

- 자 이제 잘 돌아갔는지 실행해 볼까요?
- 앞에서 구현한 모든 메소드를 main 함수에 넣고 한번 돌려보세요.

```
Print By Indexed For:
Point@15db9742 Point@6d06d69c Point@7852e922 Point@4e25154f Point@70dea4e
Print By Iterator:
Point@15db9742 Point@6d06d69c Point@7852e922 Point@4e25154f Point@70dea4e
Print By Enhanced For:
Point@15db9742 Point@6d06d69c Point@7852e922 Point@4e25154f Point@70dea4e
```

- 무엇인가 이상한 것이 나오네요?
- 다들 (x, y)같이 깔끔한 형식을 기대 하지 않았나요?

여기서 질문!

- 분명 System.out.print 함수를 통해서 우리의 객체를 출력하였습니다.
- 근데 이 함수는 콘솔 창에 문자열로 보여주는 일을 하지요.
- 우리는 그 어디에도 해당 객체가 문자열로 바뀌게 하는 코드를 구현하지 않았죠.
- 과연 우리의 객체는 어디서 String으로 변환 되었을까요?

Java에는 신이 계십니다.

- Java의 모든 객체는 Object를 상속합니다. 따라서 모든 객체를 Object로 받을 수 있으며, Object에 구현된 모든 함수를 사용합니다.
- Object 객체에는 toString이라는 함수가 있습니다. 이는 객체를 문자열로 바꿔주는 함수입니다.

- 즉 Point는 좋든 싫든 Object를 상속받고 있으며, 따라서 print 함수 안에서 Point는 Object의 toString을 호출한 것이죠.

그렇다면?

- toString의 기본 형식은
  (클래스 이름) @ (해시 코드)
  입니다.

- 그렇게 의미 있는 정보는 아니죠.
  어떻게 하면 일관성을 해치지 않고
  Point를 우리가 원하는 형식으로 출력할 수 있을까요?

- 앞 시간에 우리는 배웠습니다.
  바로 오버로딩이 아니라 오버라이드(Override)입니다.

이걸 또 Eclipse가!

- Point에 toString 함수를 직접 구현 해도 되겠지만
  우리의 Eclipse는 귀찮은 것은 자기에게 맡기라고 하네요?

- Getter, Setter 처럼 toString 함수도 자동 완성 기능을 제공해 줍니다.
- 오른쪽 클릭 > Source > Generate toString() 클릭!

```
@Override
public String toString() {
	return "[x=" + x + ", y=" + y + "]";
}
```

아마 이 형식이 아니라 다른 형식일 거예요.
이건 조금 수정한 거랍니다.

아…아름다워…

```
Print By Indexed For:
[x=0.0, y=0.0] [x=2.0, y=5.0] [x=4.0, y=4.0] [x=7.0, y=9.0] [x=4.0, y=4.0]
Print By Iterator:
[x=0.0, y=0.0] [x=2.0, y=5.0] [x=4.0, y=4.0] [x=7.0, y=9.0] [x=4.0, y=4.0]
Print By Enhanced For:
[x=0.0, y=0.0] [x=2.0, y=5.0] [x=4.0, y=4.0] [x=7.0, y=9.0] [x=4.0, y=4.0]
```

SET<T>

두 번째 자료구조

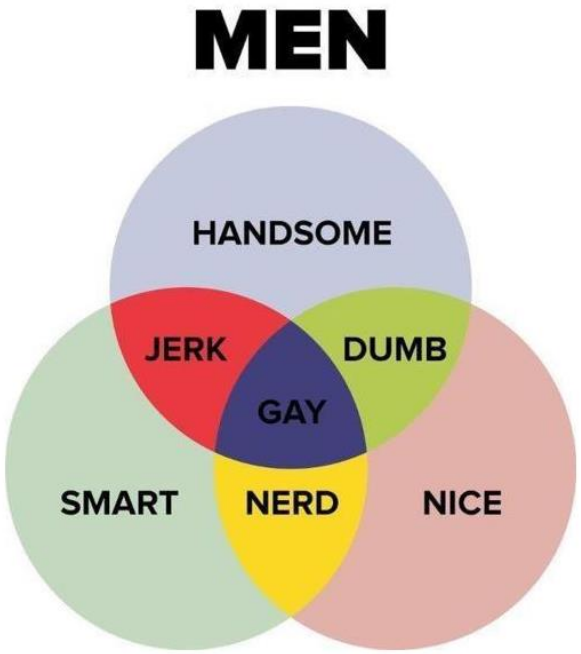
equals와 hashCode

집합?

- 이번에는 집합 자료구조를 만들어 보겠습니다.

- 먼저 집합이 뭔가요?
  모르면 고1부터…

- 네. 저도 정확한 정의는 잘 기억이 나지 않지만,
  확실한 것은 똑같은 것이 2개 이상 들어가지 못하는 것이죠.



Set<T>

- List와 비슷합니다.
  Set<T>는 인터페이스이고,
  HashSet<T>는 이를 구현한 클래스
  랍니다.

- HashSet<T>말고도 여러가지 Set 클
  래스가 많이 있습니다.

- 오른쪽과 같이 구현해 주세요.

```
Set<Point> setOfPoints = new HashSet<Point>(listOfPoints);
```

생성자가 이상해요.

```
public HashSet(Collection<? extends E> c)
```

Constructs a new set containing the elements in the specified collection. The HashMap is created with default load factor (0.75) and an initial capacity sufficient to contain the elements in the specified collection.

Parameters:

c - the collection whose elements are to be placed into this set

Throws:

NullPointerException - if the specified collection is null

- 생성자도 오버로딩을 할 수 있습니다.
- Set<T>는 List<T>를 받아서 곧바로 Set을 만들 수 있습니다.
- 그 외에도 여러가지 생성자가 있으니 필요하면 직접 검색해서 활용해 보세요.

출력을 해볼까요?

- Set은 indexed for로는 참조할 수 없습니다.
  순서대로 정리되어 있지 않다는 이유겠지요?

- 대신 Iterator와 enhanced for를 통해서 반복시킬 수 있습니다.

- List와 비슷하게 출력하는 함수를 구현해 보세요.

```
Iterator<T> iter = list.iterator();
while(iter.hasNext()) {
	System.out.print(iter.next()+" ");
}
```

```
for(T t : list) {
	System.out.print(t+" ");
}
```

이건 집합이 아니잖아?

```
Print By Iterator:
[x=4.0, y=4.0] [x=4.0, y=4.0] [x=0.0, y=0.0] [x=2.0, y=5.0] [x=7.0, y=9.0]
Print By Enhanced For:
[x=4.0, y=4.0] [x=4.0, y=4.0] [x=0.0, y=0.0] [x=2.0, y=5.0] [x=7.0, y=9.0]
```

무엇이 너희를 다르게 하였나?

- 왜 그럴까요? 사실 이 질문은 조금 어렵습니다.
  x와 y가 동일하면 당연히 똑같다고 생각할 수 있기 때문입니다.
- 하지만 이는 엄연히 '인간'의 생각입니다.
  사람은 쉽게 논리적으로 그렇게 생각할 수 있지만,
  멍청한 컴퓨터는 그것을 간파하지 못합니다.
- 컴퓨터의 입장에서는 메모리 상에 서로 다른 위치에 있는 다른 객체로 파악합니다.
  우연히 두 객체의 필드가 모두 동일한 것이죠.
- 예를 들면 아무리 모습이 비슷한 쌍둥이라도 둘은 엄연히 다른 것과 마찬가지죠.
  ~~(물론 특징이 똑같은 도플갱어를 보면 좀 무섭겠죠…?)~~

그럼 어떻게 해요?

- 그렇다고 방법이 없는 것은 아닙니다.
  컴퓨터에게 두 객체가 같다고 알려주면 된답니다.
- 여기서 또 우리의 절대 지존 Object 객체가 나옵니다.
- Java에서 두 객체가 같은 지를 비교할 때는
  Object의 equals와 hashCode 함수를 활용합니다.

- Set도 동일하게 equals와 hashCode를 사용하여 같은 지를 비교하고,
  만약 같은 것이 들어오면 과감히 내버린 답니다.
- 앞 예제의 toString 함수와 비슷하게 equals와 hashCode 함수를 오버라이드하면, 우리가 원하는 것을 이룰 수 있겠죠?

혹시…?

```java
@Override
public int hashCode() {
	final int prime = 31;
	int result = 1;
	long temp;
	temp = Double.doubleToLongBits(x);
	result = prime * result + (int) (temp ^ (temp >>> 32));
	temp = Double.doubleToLongBits(y);
	result = prime * result + (int) (temp ^ (temp >>> 32));
	return result;
}
```

```java
@Override
public boolean equals(Object obj) {
	if (this == obj)
		return true;
	if (obj == null)
		return false;
	if (getClass() != obj.getClass())
		return false;
	Point other = (Point) obj;
	if (Double.doubleToLongBits(x) != Double.doubleToLongBits(other.x))
		return false;
	if (Double.doubleToLongBits(y) != Double.doubleToLongBits(other.y))
		return false;
	return true;
}
```

- 혹시 자동 완성 기능을 기대하셨나요?
- 그렇다면 Eclipse의 활용에 조금 능숙해 지셨다는 뜻이랍니다.
- 오른쪽 클릭 > Source > Generate hashCode() and equals() 클릭!
- 깔끔하게 완성되었네요!

동작도 문제 없어요!


```
Print By Iterator:
[x=0.0, y=0.0] [x=4.0, y=4.0] [x=2.0, y=5.0] [x=7.0, y=9.0]
Print By Enhanced For:
[x=0.0, y=0.0] [x=4.0, y=4.0] [x=2.0, y=5.0] [x=7.0, y=9.0]
```

MAP<K, V>

세 번째 자료구조

지도?

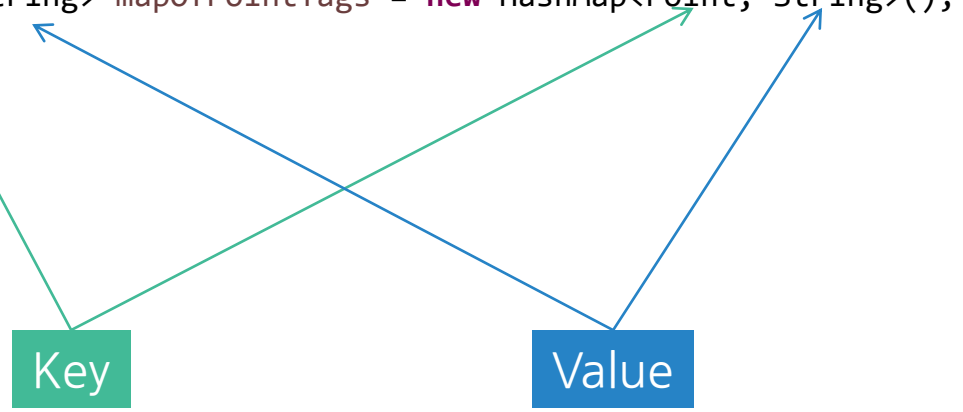
- 이번에는 조금 생소할 수도 있지만,
  정말 활용도가 높은 객체인 Map을 알아보겠습니다.
- Map은 Key와 Value가 짝을 이루고 있는 자료구조입니다.
  Key는 반드시 유일해야 합니다. 즉 Set을 이룹니다.
  Value는 유일할 필요는 없습니다. Key에 해당하는 값을 가질 뿐입니다.

사전?

- 쉬운 예로 사전이 있습니다.
  사전은 단어와 단어에 대한 해석이 적혀 있지요.
- 단어는 중복되지 않습니다. 글자가 같아도 1, 2, 3으로 구분해 놓았지요.
  단어를 Key라고 생각하시면 되겠습니다.
- 사전의 목적은 단어의 해석을 보는 것이지요.
  똑같이 Map도 Key를 통해서 해당 Value를 가지고 노는 것이 주된 목적입니다.

어떻게 쓰나?

```
Map<Point, String> mapOfPointTags = new HashMap<Point, String>();
```



- Map<K, V>는 앞의 자료구조와는 조금 다른 모습을 보입니다.
  바로 Generics가 2개 필요하다는 점인데요.
- 당연히 Key와 Value에 해당하는 객체를 둘 다 알아야 하니까 그렇겠죠?
- 이제 점 하나하나에 꼬리표를 달아줘 볼까요?

결과가 어떤가요?

- 이제 하나씩 꼬리표를 달아봅시다.
- put 함수를 사용하면 Map에 점과 문자열의 쌍을 넣겠다는 뜻입니다.
- get 함수를 사용하면 이를 꺼내는 작업을 합니다.

- 각각의 결과가 어떤가요?
  특히 마지막 결과는 무엇일까요?

```
mapOfPointTags.put(p1, "Origin");
mapOfPointTags.put(p2, "A");
mapOfPointTags.put(p3, "B");
mapOfPointTags.put(p4, "C");
mapOfPointTags.put(p5, "D");

System.out.println(mapOfPointTags.get(p1));
System.out.println(mapOfPointTags.get(new Point(2f, 5f)));
System.out.println(mapOfPointTags.get(p3));
```

정렬

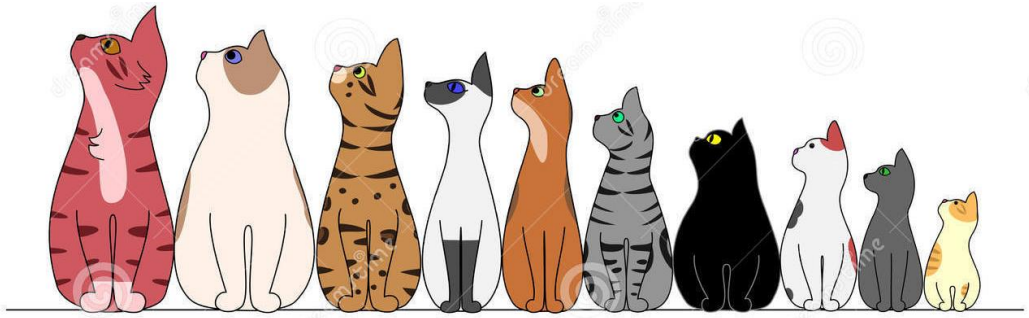
Collections.sort

Comparable과 Comparator

Anonymous Class와 Template Method Pattern

키 작은 순으로 서 봐~

- 자료구조에서, 특히 List에서 자료를 정렬하는 것은 매우 중요합니다.
  때문에 여러가지 정렬 알고리즘이 있습니다.
- 우리가 구현한 List<T>는 어떻게 정렬할까요?
  직접 뜯어서 우리가 하나씩 정렬해 주어야 할까요?
  설마요….
- Java에서는 Collections라는 객체를 통해서 여러가지 Collection에 필요한 함수들을 제공하고 있습니다.



Collections

- Collections의 sort 함수를 사용하면 List를 정렬할 수 있습니다.

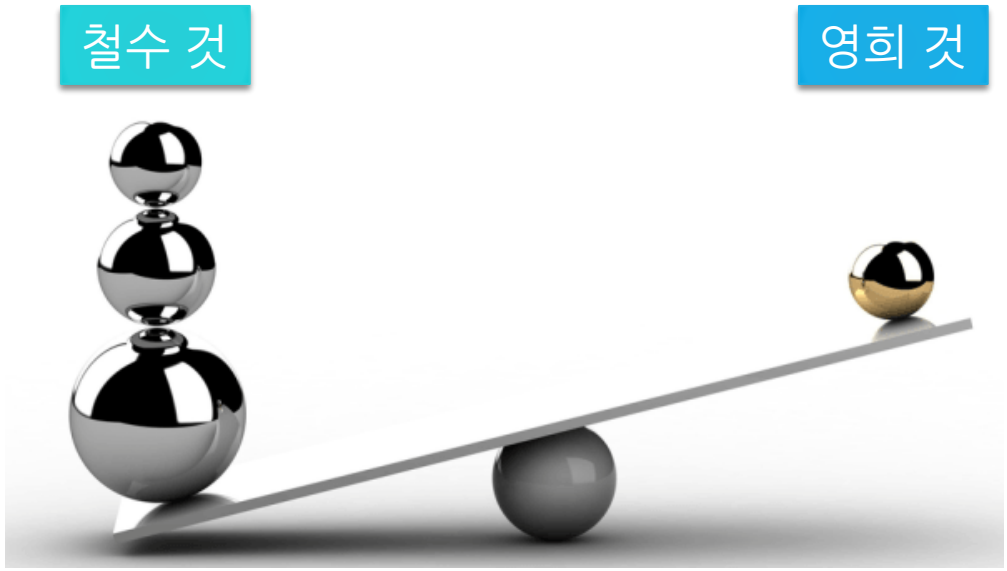
```
public static <T extends Comparable<? super T>> void sort(List<T> list)
public static <T> void sort(List<T> list, Comparator<? super T> c)
```

- 위의 것은 조금 감이 오네요. 파라미터가 하나니까, 이 list를 정렬한다는 의미겠죠.
  대신 좀 더 정확히 알기 위해서는 Comparable이 무엇인지 알아야 합니다.
- 아래는 두 번째 파라미터가 도대체 무엇인지 감이 잘 오지 않을 거에요.
  아래의 것을 이해하기 위해서는 Comparator가 무엇인지 설명해야 합니다.
- 객체 이름에 주의하세요. Collections로 끝에 s가 붙어있습니다.
  ~~(이건 Naming이 많이 구린 듯…)~~

Comparable

- Comparable은 인터페이스입니다.
- 이 인터페이스를 구현한 객체는 반드시 compareTo 함수를 구현해야 합니다.
- compareTo에서는 비교하는 객체보다 자신이 큰지 같은지 작은지를 반환합니다.

Comparator



- Comparator도 인터페이스입니다.
- 하지만 Comparable과는 다르게 비교하는 객체에 구현하지 않습니다. 대신 두 객체를 비교하는 새로운 객체에 구현합니다.
- 여기서 compare 함수를 구현하면 됩니다.

여기서만 쓸 거 같은데…

- 저는 Comparator를 활용해서 비교를 해보도록 하겠습니다.

- 근데 이번 한번만 필요할 거 같은데, 객체를 새로 생성해야 할까요
- 객체를 많이 만드는 것도 좋지만,
  그렇다고 너무 많이 만들면 복잡해지겠죠?

일회용 객체?

- 이런 고민을 하는 당신을 위한 최적의 선택!
- Java에는 객체를 필요할 때 딱 한번 쓰고 버릴 수 있는 방법이 있습니다.
  바로 익명 클래스(Anonymous Class)입니다.
- 인터페이스를 구현해야 하는데 그렇게 많이 쓰이지 않거나,
  급하게 필요한 경우에 한해서 사용하면 좋습니다.
- 익명 클래스인 이유는 해당 인터페이스를 구현하지만
  클래스 이름이 없기 때문이지요.

자 이제 정렬해 보자!

익명 클래스

```
Collections.sort(listOfPoints, new Comparator<Point>(){

	@Override
	public int compare(Point arg0, Point arg1) {
		if(arg0.getX() < arg1.getX())
			return -1;
		else if(arg0.getX() > arg1.getX())
			return 1;
		else
			return 0;
	}

});
```

- x좌표를 기준으로 오름차순으로 정렬시키고자 합니다.
- Sort 함수는 Comparator의 규칙에 따라 List를 오름차순(작은 것에서 큰 것)으로 정렬합니다.
- Compare 함수가
  음수를 반환하면 앞의 것이 작다는 뜻입니다.
  양수를 반환하면 앞의 것이 크다는 의미입니다.
  당연히 0을 반환하면 같다는 의미입니다.

드디어!

```
Print By Enhanced For:
[x=0.0, y=0.0] [x=2.0, y=5.0] [x=4.0, y=4.0] [x=4.0, y=4.0] [x=7.0, y=9.0]
```

연습문제

1. List<T>와 Set<T>를 동시에 정렬하는 메소드를 만들어 보세요.
2. Y 필드에 대해서 내림차순으로 정렬해 보세요.
3. Comparable 인터페이스를 활용하여 정렬해 보세요.
4. 이번엔 점들의 집합으로 이루어진 다각형(Polygon)이라는 객체를 생각해 봅시다. 주어진 소스 코드를 완성시켜 보세요!

수고하셨습니다

질문이 있으면 말씀해 주세요!