Problems with ParkingLot / ListSL Design and Implementation

• Lack of clean separation between application specific code (ParkingLot) and generally useful utilities (ListSL).
• Class casts are needed because the Java compiler does not know the types of data being stored in a ListSL.
Interface

- Similar to a class.
- Includes declarations of methods.
- Does not include implementations of methods.
- A class may be declared to implement an interface.
- The name of the interface can be used in contexts where class names are used.
List Interface

public interface List {
    void add(Object data);
    Object remove (Object data);
    boolean contains(Object data);
    int size();
    boolean isEmpty();
    void clear();
    String toString();
}

The List interface defines the signatures of operations on lists without specifying how they are implemented. The List interface may be used in contexts where a data type is needed.
Using the List Interface

ParkingLot Project

```java
public class ListSL { // Omitted }
```

Working Project

```java
public class ListSL implements List { // Omitted }
```

In the `ListSL` class definition, we declare that `ListSL` implements `List`. The Java compiler will require us to implement all the methods of the `List` interface (along with any other methods we want).
Using the List Interface

ParkingLot Project

```
ListSL free = new ListSL();
```

Diagram:

↓

Working Project

```
List free = new ListSL();
```

The **ListSL** class name is used only with the **new** operator, to indicate the class we choose to implement the **List** interface. In all other contexts we use the **List** interface name. Now if we want to substitute **ListDL** for **ListSL**, we need only change a few locations.
Generic Classes

```java
public class ListSL<D> implements List<D> {

    // Omitted ...
    public void add(D data) { // Omitted ... }  // Omitted ...
    public D remove(D data) { // Omitted ... }  // Omitted ...

}

class ListElementDL<D> {

    // Omitted ...
    D data() { // Omitted ... }  // Omitted ...

}
```

The `ListSL` and `ListElementSL` classes are declared to be generic. A `ListSL` and `ListElementSL` can be specialized by supplying a concrete type for the type parameter D in the declaration.
Generic List Interface

public interface List<D> {
    void add(D data);
    D remove(D data);
    boolean contains(D data);
    int size();
    boolean isEmpty();
    void clear();
    String toString();
}

The same as the earlier List interface, but parameterized with the data type D, which replaces Object.
Using Generic Classes

Working Project

List free = new ListSL();

ParkingLotGeneric Project

List<ParkingSpace> free = new ListSL<ParkingSpace>();

Now the Java compiler will allow only ParkingSpace objects to be added to the free list. Also, the Java compiler knows that anything removed from the free list is a ParkingSpace object.
Removing Class Casts

ParkingLot Project

```java
ParkingSpace location = (ParkingSpace) free.remove(request);
```

ParkingLotGeneric Project

```java
ParkingSpace location = free.remove(request);
```

No possibility of a class cast exception! (Why not?)
Generics with Type Restrictions

- Sometimes a generic container needs to perform computations on its contents.
- E.g., comparing elements to see which is larger.
- E.g., adding elements to compute an arithmetic sum.
- The concrete type must support these operations.
- How can we enforce this restriction?
public class Association<K extends Comparable<K>, D> {

    protected K key;
    protected D data;

    // Omitted ...

    public int compareTo(Association<K,D> other) {
        return key().compareTo(other.key());
    }

    // Omitted ...

}
**Generic Comparable Interface**

```java
public interface Comparable<T> {
    int compareTo(T other);
}
```

A class implements the Comparable interface if it implements the `compareTo` method.
Generic Methods

• A method may be declared generic, even if the class in which it’s defined is not.
• The Java compiler can often figure out the concrete type needed when the method is invoked.
• Sometimes the compiler needs a hint.
public class MinMaxGeneric {

    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x;
        if (y.compareTo(max) > 0) {
            max = y;
        }
        if (z.compareTo(max) > 0) {
            max = z;
        }
        return max;
    }

    public static void main(String args[]) {
        System.out.printf("Max of %d, %d and %d is %d%n", 3, 4, 5, 
            maximum(3, 4, 5));

        System.out.printf("Max of %.1f,%.1f and %.1f is %.1f%n", 6.6, 8.8, 7.7, 
            maximum(6.6, 8.8, 7.7));

        System.out.printf("Max of %s, %s and %s is %s%n", "pear", "apple", "orange", 
            maximum("pear", "apple", "orange"));
    }

}