Another design pattern for this can be prototype design pattern:

```
public abstract class Vehicle implements Cloneable {

   private String id;
   protected String type;

   abstract int set_num_of_wheels();
   abstract int set_num_of_passengers();
   abstract boolean has_gas();


   public String getType(){
      return type;
   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public Object clone() {
     Object clone = null;

      try {
         clone = super.clone();

      } catch (CloneNotSupportedException e) {
         e.printStackTrace();
      }

      return clone;
   }
}

public class Cars extends Vehicle {

   public Cars(){
     type = "Cars";
   }

   @Override
   public abstract int set_num_of_wheels(){};
   public int set_num_of_passengers(){};
   public boolean has_gas(){};
}
```

```
public class Planes extends Vehicle {

   public Planes(){
     type = "Planes";
   }

   @Override
   public abstract int set_num_of_wheels(){};
   public int set_num_of_passengers(){};
   public boolean has_gas(){};
}
```

```
import java.util.Hashtable;

public class VehicleCache {

   private static Hashtable<String, Vehicle> vehicleMap  = new
Hashtable<String, Vehicle>();

   public static Vehicle getVehicle(String vehicleId) {
      Vehicle cachedVehicle = vehicleMap.get(vehicleId);
      return (Vehicle) cachedVehicle.clone();
   }

   public static void loadCache() {
      Cars car = new Cars();
      car.setId("1");
      vehicleMap.put(car.getId(),car);

      Planes plane = new Planes();
      plane.setId("2");
      vehicleMap.put(plane.getId(),plane);
   }
}
```

```
public class PrototypePattern{
   public static void main(String[] args) {
      VehicleCache.loadCache();

      Vehicle clonedVehicle =(Vehicle)VehicleCache.getVehicle("1");
      System.out.println(clonedVehicle.getType());

      Vehicle clonedVehicle2=(Vehicle)VehicleCache.getVehicle("2");
```

```
        System.out.println(clonedVehicle2.getType());
}
```