Entidades

```
public class Bike {
  private Long bikeId;
  private String name;
  private String description;
  private double dayPrice;
  private int units;
  private LocalDateTime creationDate;
  private long numberOfRatings;
  private double avgRating;

  // Constructores, getters/setters, equals, hashCode

}

public class Reservation {
 private Long reservationId;
 private String userEmail;
 private Long bikeId;
 private String creditCardNumber;
 private LocalDate startDate;
 private LocalDate endDate;
 private byte units;
 private LocalDateTime reservationDate;
 private byte rating;     // Alternativamente se podría modelar como float/double
                          // Se podría usar el valor -1 para indicar que aún no tiene puntuación o
                          //   tener un atributo adicional ‘rated’ para indicar si se puntuó o no

  // Constructores, getters/setters, equals, hashCode

}
```

Servicio del modelo

```
public interface BikeService {

   // Validaciones InputValidationException:
   //      - bike.name == null || bike.name.length() == 0
   //      - bike.description == null || bike.description .length() == 0
   //      - bike.units, bike.dayPrice <= 0
  public Bike addBike(Bike bike) throws InputValidationException;


   // Validaciones InputValidationException:
   //      - Mismas que en addBike
   // Validaciones BikeWithReservationsException
   //      - Si tiene reservas
   public void updateBike(Bike bike) throws InstanceNotFoundException,
         InputValidationException, BikeWithReservationsException;


   public List<Bike> findBikes(String keywords);


   // Validaciones InputValidationException:
   //      - email == null || email no sigue el patrón user@domain
   //      - creditCardNumber: no formado por 16 dígitos numéricos
   //      - units <= 0
   //      - endDate < startDate
   //      - endDate - startDate > 14 días
   //      - startDate <= día actual
   // Validaciones NotEnoughUnitsException
   //      - units > dbBike.units
   public Reservation reserveBike(String userEmail, Long bikeId, String creditCardNumber,
         LocalDate startDate, LocalDate endDate, int units) throws
         InstanceNotFoundException, InputValidationException, NotEnoughUnitsException;


   // Validaciones InputValidationException:
   //      - userEmail == null || userEmail no sigue el patrón user@domain
   public List<Reservation> findUserReservations(String userEmail) throws
         InputValidationException;
```

```
    // Validaciones InputValidationException:
    //      - userEmail == null || userEmail no sigue el patrón user@domain
    //      - points < 0 || points > 10
    // Validaciones AlreadyRatedException
    //      - Si no existe el atributo ‘rated’: dbReservation.rating >= 0
    //      - Si existe el atributo ‘rated’: dbReservation.rated == true
    // Validaciones ReservationNotFinishedException
    //      - dia actual <= dbReservation.endDate
    // Validaciones IncorrectUserException
    //      - userEmail != dbReservation.userEmail
    public void rateReservation(Long reservationId, String userEmail, byte points)
            throws InstanceNotFoundException, InputValidationException,
            AlreadyRatedException, ReservationNotFinishedException, IncorrectUserException;

}
```

DAOs

```
public interface SqlBikeDao {

	public Bike create(Connection connection, Bike bike);

	public void update(Connection connection, Bike bike)
		throws InstanceNotFoundException;

	// Necesario para updateBike y reserveBike
	public Bike find(Connection connection, Long bikeId)
		throws InstanceNotFoundException;

	public List<Bike> findBikes(Connection connection,
		String keywords);

	// Necesario para las pruebas
	public void remove(Connection connection, Long bikeId)
		throws InstanceNotFoundException;
}

public interface SqlReservationDao {

	public Reservation create(Connection connection, Reservation reservation);

	// Necesario para rateReservation
	public void update(Connection connection, Reservation reservation)
		throws InstanceNotFoundException;

	// Necesario para rateReservation
	public Reservation find(Connection connection, Long reservationId) throws
		InstanceNotFoundException;

	// Necesario para updateBike (para saber si ya hay reservas para un modelo de bicicleta)
	public boolean existsAnyReservation(Connection connection, Long bikeId);

	public List<Reservation> findByUser(Connection connection, String userEmail);

	// Necesario para las pruebas
	public void remove(Connection connection, Long reservationId)
		throws InstanceNotFoundException;

}
```