```
/**
 * The class Winery implements the business layer.
 *
 * @author
 * @version 1.0
 */

public class Winery {

	private static EntityManagerFactory emf;
	private static EntityManager em;
	private static final Logger logger = Logger.getLogger(Winery.class.getName());
	private static Winery instance;
	private List<Wine> wines = new ArrayList<Wine>();

	private List<Producer> producerList = new ArrayList<Producer>();

	private Winery() {
		emf = Persistence.createEntityManagerFactory("myWinery");
		em = emf.createEntityManager();
	}

	@SuppressWarnings("unchecked")
	public synchronized List<Wine> searchCatalogWines(String name,
			String country, String color) throws MissingSearchCriteriaException {
		logger.info("Search wines");
		if (name.isEmpty() && country.isEmpty() && color.isEmpty()) {
			throw new MissingSearchCriteriaException();
		}

		Query query =
em.createNamedQuery(Wine.FIND_BY_NAME_COUNTRY_COLOR);
		query.setParameter("name", "%" + name + "%");
		query.setParameter("country", "%" + country + "%");
		query.setParameter("color", "%" + color + "%");

		List<Wine> results = (List<Wine>) query.getResultList();
		for (Wine wine : wines) {
			if (wine.getName().toLowerCase().indexOf(name) >= 0
				&& wine.getCountry().toLowerCase().indexOf(country) >= 0
					&& wine.getColor().toLowerCase().indexOf(color) >= 0) {
				results.add(clone(wine));
			}
		}
		return results;
	}

	public void addWine(Wine wine){
		em.getTransaction().begin();
		em.persist(wine);
		em.getTransaction().commit();
	}
	public void addProducer(Producer producer){
```

```
		em.getTransaction().begin();
		em.persist(producer);
		em.getTransaction().commit();
	}
	public void updateWine(Wine wine){
		em.getTransaction().begin();
		em.merge(wine);
		em.getTransaction().commit();
	}

	public String deleteWine(Wine wine) {
		em.getTransaction().begin();
		em.remove(wine);
		em.getTransaction().commit();

		return null;
	}
	public String deleteProducer(Producer producer) {
		em.getTransaction().begin();
		em.remove(producer);
		em.getTransaction().commit();

		return null;
	}
	public void updateProducer(Producer producer){
		em.getTransaction().begin();
		em.merge(producer);
		em.getTransaction().commit();
	}

	@SuppressWarnings("unchecked")
	private <T> T clone(T object) {
		try {
			ByteArrayOutputStream os = new ByteArrayOutputStream();
			new ObjectOutputStream(os).writeObject(object);
			ByteArrayInputStream is = new
ByteArrayInputStream(os.toByteArray());
			return (T) new ObjectInputStream(is).readObject();
		} catch (Exception e) {
			logger.severe(e.toString());
			return object;
		}
	}

	public static Winery getInstance() {
		if (instance == null) {
			instance = new Winery();
		}
		return instance;
	}

	public List<Wine> getWines() {
		TypedQuery<Wine> query = em.createQuery("select w from Wine w",
```

```
Wine.class);
		return query.getResultList();
	}

	public Wine getWineById(long id){

TypedQuery<Wine> query = em.createQuery("select w from Wine w where id=" +
Long.toString(id), Wine.class);
			return query.getSingleResult();
	}
	public List<Producer> getProducers() {
		TypedQuery<Producer> query = em.createQuery("select p from Producer
p", Producer.class);
		return query.getResultList();
	}

	public List<Producer> getProducersById(long pid)
	{
		TypedQuery<Producer> query = em.createQuery("select p from Producer p
where p.id=" + pid, Producer.class);
		return query.getResultList();
	}
	public List<Wine> getWinesName(String name) {
		TypedQuery<Wine> query = em.createQuery("select w from Wine w where
name LIKE :name", Wine.class);
		query.setParameter("name", "%" + name + "%");
		return query.getResultList();
	}
	public List<Wine> getWinesCountry(String country) {
		TypedQuery<Wine> query = em.createQuery("select w from Wine w where
country LIKE :country", Wine.class);
		query.setParameter("country", "%" + country + "%");
		return query.getResultList();
	}
	public List<Producer> getProducersName(String name) {
		TypedQuery<Producer> query = em.createQuery("select p from Producer p
where name LIKE :name", Producer.class);
		query.setParameter("name", "%" + name + "%");
		return query.getResultList();
	}
	// Converter

	public List<Producer>getAllProducers(){
		return producerList;
	}

	public Producer findProducer(long producerId){
		for(Producer producer: producerList){
			if(producer.getId() == producerId){
				return producer;
			}
		}
		return null;   }     }
```