**1.**

**a)** Genus is the parent class of species
**b)** Specimen contains instance of species (species is nested inside specimen)
**c)**

**d)** Programming team won't need to copy and paste the characteristic since the lower level taxonomy can copy the characteristic of its parent by using OOP and the characteristics are grouped under one object. (the data is in one pack)
**e)** Because toString() is overrided in species class and even though it is not overrided, Java would look it up from its parent until it reached the very base of it.

**2**

**a)** Information within a class will be secured within that class scope and not to be accessed freely from outside the class unless it is neccesary or meant to be.
**b)** Data inside won't be mistakenly or unconsciously changed which may lead to problems and make sure that data is not mixed up with those outside the scope.
**c)** specimen.getName()
**d)** name<String>
**e)**

```
public class Genus {
	private String genusName;

	public Genus(String genusName) {
		this.genusName = genusName;
	}

	public void setGenusName(String genusName) {
		this.genusName = genusName;
	}

	public String getGenusName() {
		return this.genusName;
	}

	@Override
	public String toString() {
		return "Genus: " + this.genusName;
	}
}
```

**f)** Advantage: We can use method or properties from its parent (Species and Genus)
Disadvantage: Break encapsulation

**3.**

**a)**

```
public class Specimen {
	private String name;
	private  int cageNumber;
	private Species toa;
	private String description;
	public Specimen (String a, int c, species s, String description) {
		this.name = name;
		this.cageNumber = c;
		this.species = s;
		this.description = description;
	}

	public void setName (String a) {this.name = a;}
	public void setCage (int c) {this.cageNumber = c;}
	public void setTOA (Species s) {this.toa = s;}
	public void setDescription (String description) {this.description = description;}
	public String getName () {return this.name;}
	public int getCage() {return this.cageNumber;}
	public Species getTOA() {return toa;}
	public String getDescription() {return this.description;}
	public String toString() {return name + " is a " + toa + " in cage " +
cageNumber;}
}
```

**b)**

```
public static int countSpecimens ( Specimen[] animals, Species s) {
	int numOfSpecies = 0;
		for (Specimen animal : animals) {
			if (animal.getTOA().equals(s)) {
				numOfSpecies += 1;
			}
		}
	return numOfSpecies;
}
```

**c)**

```
public static LinkedList<String> listSpecies (Specimen[] animals) {
	LinkedList<String> list = new LinkedList<String>();
	for (Specimen animal : animals) {
		if (!list.contains(animal.getTOA().getSpeciesName())) {
			list.add(animal.getTOA().getSpeciesName());
		}
	}
	return list;
}
```

**4.**

**a)** ADT shows how an object should look like (what function it serves) but not specifically the implementation

**b)**

```
public static LinkedList<Specimen> makeList (Specimen[] animals) {
	return new LinkedList<Specimen>(Arrays.asList(animals));
}
```

**c)**

```
public static LinkedList<Species> makeSpeciesList (LinkedList<Specimen> animals) {
	LinkedList<Species> list = new LinkedList<Species>();
	for (Specimen animal: animals) {
		list.add(animal.getTOA());
	}
	return list;
}
```

**d)**

```
public static LinkedList<Species> makeSpecialSpeciesList (LinkedList<Species>
animals) {
	LinkedList<Species> list = new LinkedList<Species>();
	boolean isExist;

	for (Species animal: animals) {
		isExist = false;
		for (Species data : list) {
			if (animal.getSpeciesName().equals(data.getSpeciesName())) {
				isExist = true;
				break;
			}
		}

		if (!isExist) {
			list.add(animal);
		}
	}
	return list;
}
```