Appendix A

Experimental Artifacts

This appendix contains the artifacts used in our two studies that could not be detailed in our papers. They were translated from their originals in Portuguese.

A.1 Study One

A.1.1 Subject Guide

Subject Guide I

Hello!

The goal of this experiment is to evaluate how conflicts between concurrent tasks are managed by software developers working as a team.

```
> Please, turn off or silence you mobile phone <
```

You will start by watching a video describing the views that you will use in Eclipse, taking as example a programming situation similar to that in this experiment.

```
> Press ▶ and watch the video with attention <
```

You have just watched a programmer using Eclipse to manage conflicts between his tasks and those of a co-worker. During this experiment you (Bob) and your co-worker (Anne) will modify an application at the same time, and be watched on how you manage conflicts.

```
> Go to Eclipse (use CMD + TAB) <
```

Inside Eclipse you will find the code of the Zoo application, whose concepts are illustrated in the figure below. Succinctly, a Zoo has zones (Zone) each one occupying its area (Area). Each zone has animals of a single species (the non-abstract classes, e.g., Snake), and the animals get energy by eating (Food). The menus have options for managing the Zoo — e.g., to manage zones, allocate animals to zones, and sell tickets.

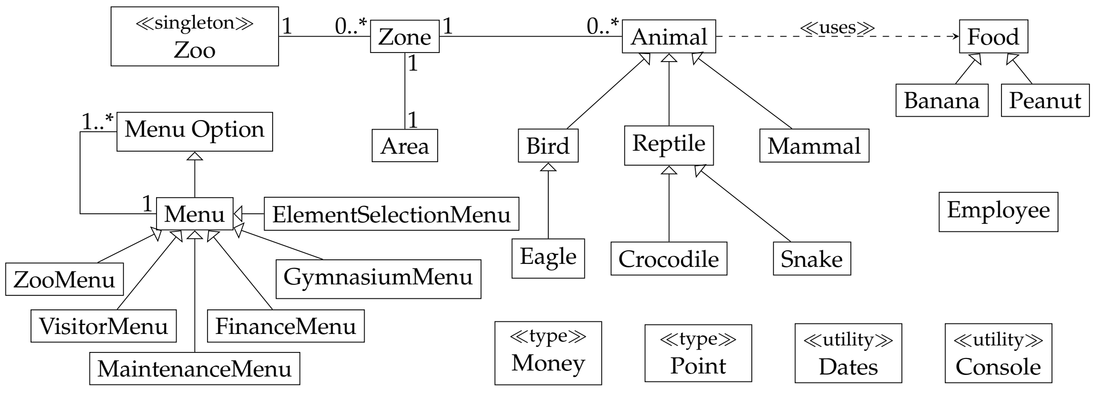

Figure 1: Conceptual class diagram.

Let's continue!

A video will be recorded during the experiment with everything you do in the computer. You must say out loud your actions, observations, and the rationale of the decisions you take during your tasks, like when you start or end a task, you see a conflict, you decide some action to resolve a conflict, you see something interesting in a view, etc.

Privacy Statement
The video is for scientific use only, and all public conclusions will be anonymous. Please, do not disclose the contents of this experiment to not affect this research.

Soon a list of tasks will be given to you with the code that you have to type in each task. You should read the code to understand what the task means, instead of just typing it. This is important so that you can detect conflicts and resolve them correctly.

During your tasks you must respect the following points:

- Type the shaded code given in the task (you may format it at will);
- Save every time you finish modifying a file;
- Fix all compilation errors before moving to new tasks;
- Do the tasks in the presented order (do not jump tasks);
- Resolve the conflicts, thinking on the best resolutions for your team, even if that means you need to revert your changes or rewrite the code. If you need to negotiate a resolution you may talk to Anne via the chat;
- You will check in only when the guide tells you.

Are you ready? This is a good time for clarifications. Please do so if you need! Let's start!

```
> Do SHIFT + CMD + 2 <
```

You should see a window with a red button named "REC".

```
> Press ``REC'', and when the counting ends you may begin
working on the tasks listed next <
```

Task List

Task 1) Add more food. In package "zoo.food" create file Rice.java with the following code:

```java
package zoo.food;
public class Rice extends Food {
  public int getKCalPerKg() {
    return 1062;
  }
}
```

Task 2) In package "zoo.animals" expand the mammals hierarchy by creating the files Primate.java, Chimpanzee.java, and Gorilla.java.

Primate.java:

```java
package zoo.animals;
import zoo.utils.Point;
public abstract class Primate extends Mammal {
  private int x = 0;
  private int y = 0;
  private void setXY(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public final Point getXY() {
    return new Point(x,y);
  }
  public Point move(int x, int y) {
    setXY(x, y);
    return getXY();
  }
  public String toString() {
    return getClass().getSimpleName() + "at point "+ getXY();
  }
}
```

Chimpanzee.java:

```java
package zoo.animals;
public class Chimpanzee extends Primate {
  public void jump(int height) {
    System.out.println("Jumping "+ height + "meters");
  }
}
```

Gorilla.java:

```java
package zoo.animals;
import zoo.food.Food;
import zoo.food.Banana;
public class Gorilla extends Primate {
  public boolean eat(Food item) {
    if (item instanceof Banana) {
      return super.eat(item);
    } else {
      System.out.println("I only like bananas!");
      return false;
    }
  }
}
```

Task 3) In package “zoo.animals” expand the mammals hierarchy by creating the files Feline.java and Lion.java.

Feline.java:

```
package zoo.animals;
import java.util.Date;
import zoo.utils.Dates;
import zoo.utils.Money;
public abstract class Feline extends Mammal {
  public Money tax(Date start, Date end) {
    long days = Dates.daysBetween(start, end);
    return getDailyTax().times(days);
  }
  protected abstract Money getDailyTax();
}
```

Lion.java:

```
package zoo.animals;
import zoo.utils.Money;
public class Lion extends Feline {
  protected Money getDailyTax() {
    return new Money("1.00");
  }
  public void roar() {
    System.out.println("GGGRRR!!! ");
  }
}
```

Task 4) In package “zoo.ui” add the new option “Pay Feline Taxes” in file FinanceMenu.java, and limit the maximum number of tickets sold in one sell to 5 in ZooMenu.java.

FinanceMenu.java:

```
import java.util.Date;
import zoo.Zone;
import zoo.animals.Feline;
...
class FinanceMenu extends Menu {
...
private static class FelineTaxOption extends MenuOption {
  public FelineTaxOption() {
    super("Pay Feline Taxes");
  }
  public void run() {
    Zone[] felineZones =
 Zoo.INSTANCE.getZonesOfFamily(Feline.class);
    if (felineZones.length == 0) {
      System.out.println("There are no felines");!
      return;
    }
    Date start = Console.readDate("Start date of tax payment");
    Date end = Console.readDate("End date of tax payment");
    Money total = new Money("0.00");
    for (Zone zone : felineZones) {
      for (Object feline : zone.getAnimals()) {
        Money tax = ((Feline)feline).tax(start, end);
        total = total.plus(tax);
      }
    }
    System.out.println("The total tax is "+ total);
  }
}
FinanceMenu() {
   super("Finance department", new BalanceOption(),
     new TicketPriceOption(),
     new DonateOption() , new FelineTaxOption());
  }
}
```

ZooMenu.java:

Add the following constant in class ZooMenu:

```
private static final int MAX_TICKETS = 5;
```

Modify method run() in class BuyTicketsOption:

```
int numTickets = Console.readInteger("How many tickets? (max "
      + MAX_TICKETS + ")");
if (numTickets > 0 && numTickets <= MAX_TICKETS) {
     ...
} else {
     System.out.println("Type a number between 1 and "
             + MAX_TICKETS + " next time :-|");
```

Task 5) The Employee class seems that is not being used, hence delete the file Employee.java in package “zoo”.

Task 6) In Area.java, in package “zoo”, make the fields “sw” (south west) and “ne” (north east) “final” — to prevent uncontrolled changes from the outside — and add the method below to compute the area.

Area.java:

```
final Point sw;
final Point ne;

public long computeArea() {
  return (ne.x - sw.x) * (ne.y - sw.y);
}
```

Task 7) If you have left any conflict to resolve, resolve it now.

Task 8) Now, you will synchronize and resolve all conflicts with the repository. Use the reference sheet of “merge actions” to help yourself in this process.

Compilation errors may occur during or after conflict resolution. That is normal, and you must resolve those errors.

```
   > If the ''Synchronize'' view is not open, go to the
''Window'' menu, choose ''Show View''->''Other...'', type
''sy'', and double-click to open the ''Synchronize'' view <
```

```
   > In the ''Synchronize'' view select the root ''Zoo'', and
in its context menu (CTRL + 🖱) choose ''Team'' and then
''Synchronize with Repository'' (wait for the operation to
finish) <
```

Remember that you should analyze concurrent changes to make the most satisfactory resolution for the whole team!

You may start now, and when you finish — without compilation errors (ignore the warnings) and “red” icons in the “Synchronize” view — move on to the next task.

Task 9) You will now update your workspace with the repository, and check in.

```
   > In the ''Synchronize'' view select the root ''Zoo'', and
in its context menu (CTRL + 🖱) choose ''Team'' and then
''Update to Head'' (wait for the operation to finish) <
```

Is everything OK? If not, resolve what is left and then move to check in.

```
   > In the ''Synchronize'' view, select the root ''Zoo'', and
in the context menu (CTRL + 🖱) choose ''Commit...'', add the
comment ''Bob's check-in'', and press ''OK'' (wait for the
operation to finish) <
```

Task 10) Congratulations, you have finished all tasks. You may stop the video recording.

```
   > Do ALT + CMD + 2 <
```

You will now fill an exit questionnaire. THANK YOU!

A.1.2 Confederate Guide

This guide will provoke the following conflicts with the tasks of the subject in the previous guide:

- Subject adds subclass `Rice` of class `Food`, and the confederate adds constructor `Food.Food(int weight)` (undefined constructor conflict — an indirect conflict);
- Subject adds new subclasses of `Mammal`, and the confederate adds constructor `Mammal.Mammal(Date birth)` (undefined constructor conflict — an indirect conflict);
- Subject adds `Primate.move(int x, int y)`, and the confederate adds `Animal.move(int deltax, int deltay)` (method override conflict — an indirect conflict);
- Subject adds new concrete subclasses of `Animal`, and the confederate adds the static method `Animal.getPrice()` (missing method conflict — an indirect conflict);
- Subject and confederate modify the body of `FinanceMenu.FinanceMenu()` (attribute change & change conflict — a direct conflict);
- Subject deletes class `Employee` and the confederate modifies class `Employee` (element change & delete conflict — a direct conflict);
- Subject and confederate modify the same fields in class `Area` (pseudo-direct conflict — a direct conflict).

The commands in the confederate guide, denoted as `> command <`, are meant to publish (RTI and WA modes) or to check in (CI mode) the code typed by the confederate, after the previous `> command <`, to provoke the above conflicts.

Guide Confederate I

Console.java:

```
public static Point readDistance() {
  return Point.parsePoint(readLine("Distance? "
    + "(given as \"(dx,dy)\" where dx and dy are integers) > ")
    .trim());
}
```

```
   > When Bob is in the middle of writing Rice:  PUBLISH /
CHECK-IN ``Added utility function.''  <
```

Animal.java:

```
private int energy = 100;
```

Food.java:

```
public abstract class Food {
  private final int weight;

  public Food(int weight) {
    this.weight = weight;
  }

  public final int getWeight() {
    return weight;
  }
  ...
}
```

Peanut.java:

```
public class Peanut extends Food {
  public Peanut(int weight) {
    super(weight);
  }
  ...
}
```

Banana.java:

```
public class Banana extends Food {
  public Banana(int weight) {
    super(weight);
  }
  ...
}
```

```
   > When Bob finishes Rice:  PUBLISH / CHECK-IN ``Some
changes on animal's energy, and the foods.''  <
```

Mammal.java:

```
public abstract class Mammal extends Animal {
  private Date birth;

  Mammal(Date birth) {
    assert birth != null;
    this.birth = birth;
  }

  public final Date getBirth() {
    return birth;
  }

  public final int getAge() {
    return Dates.yearsBetween(birth, new Date());
  }
  ...
}
```

```
   > When Bob begins Primate:  PUBLISH / CHECK-IN ``Added
birth date.''  <
```

Animal.java:

```
private Point loc = new Point(0,0);

public final Point getLoc() {
  return loc;
}

protected final void setLoc(Point loc) {
    this.loc = loc;
}

public Point move(int deltax, int deltay) {
  Zone zone = Zoo.INSTANCE.getZone(getSpecies());
  Point p = new Point(loc.x + deltax, loc.y + deltay);
  if (zone.isInside(p)) {
    loc = p;
  }
  return loc;
}

public static Money getPrice(Class<?> species) {
  assert Zoo.isConcreteAnimal(species);
```

```java
  try {
    Method getPrice = species.getMethod("getPrice");
    return (Money) getPrice.invoke(null);
  } catch (Exception e) {
    e.printStackTrace();
  }
  return null;
}

public String toString() {
  return getClass().getSimpleName()
             + "[id="+ getRegistryID()
             + ", energy="+ getEnergy()
             + ", location="+ getLoc()
             + "]";
}
```

Crocodile.java:

```java
public static Money getPrice() {
   return new Money("100.00");
}
```

Snake.java:

```java
public static Money getPrice() {
   return new Money("200.00");
}
```

Eagle.java:

```java
public static Money getPrice() {
   return new Money("300.00");
}
```

```
   > When Bob finishes Primate:  PUBLISH / CHECK-IN ``Some
more changes to animals.''  <
```

MaintenanceMenu.java (put inside the try … catch block in AddAnimalOption):

```java
Money price = Animal.getPrice(species);
if (price == null) {
  System.out.println("Sorry, price not set for "
        + zone.getSpeciesName());
  return;
}
```

```
Money balance = Zoo.INSTANCE.getBalance();
if (price.gt(balance)) {
  System.out.println("Sorry, you need "+ price + "to buy a "
        + zone.getSpeciesName());
  return;
}
zone.addAnimal((Animal)species.newInstance());
System.out.println("You added a "
        + zone.getSpeciesName().toLowerCase());
Zoo.INSTANCE.setBalance(balance.minus(price));
System.out.println("The new balance is "
        + Zoo.INSTANCE.getBalance());
```

GymnasiumMenu.java:

```
private static class RunDistanceOption extends MenuOption {
  RunDistanceOption() {
    super("Run Distance");
  }

  @Override
  public void run() {
    //Select zone
    Zone[] zones = Zoo.INSTANCE.getNonEmptyZones();
    if (zones.length == 0) {
      System.out.println("There are no animals in the zoo!");
      return;
    }
    ElementSelectionMenu menu = new ZoneSelectionMenu(zones);
    menu.run();
    Zone zone = (Zone) menu.getSelection();
    if (zone == null) return;
    //Select animal
    Object[] animals = zone.getAnimals();
    menu = new ElementSelectionMenu("Select animal", animals) {
      @Override
      protected String getLabel(Object elem) {
        return ((Animal)elem).toString();
      }
    };
    menu.run();
    Animal animal = (Animal) menu.getSelection();
    if (animal == null) return;

    //Select distance
    Point distance = Console.readDistance();
    Point destination = animal.move(distance.x, distance.y);
    if (destination != null) {
      System.out.println("The "
                         + animal.getSpeciesName().toLowerCase()
                         + "ran to "+ animal.getLoc());
    } else {
      System.out.println(
                 "The animal cannot run outside its zone!");
    }
  }
}

public GymnasiumMenu() {
  super("Animal Gymnasium", new RunDistanceOption());
}
```

```
   > When Bob is in the middle of writing Feline and Lion:
PUBLISH / CHECK-IN ``Menu stuff changed.''  <
```

ZooTests.java:

```
private static final Money MONEY_ZERO = new Money("0.00");

public void testAnimalGetPrice() {
  for (Class species : Zoo.SPECIES) {
    Money price = Animal.getPrice(species);
    String name = species.getSimpleName();
    assertNotNull("Price not defined for "+ name.toLowerCase()
        + "s. Every concrete animal class must define "
        + "a public static Money getPrice() method.", price);
    assertTrue("The price of "+ name + "s must be higher than "
        + MONEY_ZERO, price.gt(MONEY_ZERO));
  }
}
```

```
   > When Bob finishes Lion:  PUBLISH / CHECK-IN ``Changes in
tests.''  <
```

Zoo.java:

```
private Set<Employee> employees = new HashSet<Employee>();

public void addEmployee(Employee employee) {
  employees.add(employee);
}
```

```java
public void removeEmployee(Employee employee) {
  employees.remove(employee);
}

public Employee[] getEmployees() {
  return employees.toArray(new Employee[employees.size()]);
}
```

Employee.java:

```java
private final String name;
private final Date birth;

public Employee(String name, Date birth) {
  assert name != null;
  assert birth != null;
  this.name = name;
  this.birth = birth;
}

public final String getName() {
  return name;
}

public final Date getBirth() {
  return birth;
}

public final int getAge() {
  return Dates.yearsBetween(birth, new Date());
}

public String toString() {
  return "Employee [name="+ name + ", age="+ getAge() + "]";
}
```

FinanceMenu.java:

```java
private static class HireEmployeeOption extends MenuOption {
  public HireEmployeeOption() {
    super("Hire Employee");
  }

  @Override
  public void run() {
    System.out.println("Please, answer the following
        question about the new employee.");
    String name = Console.readLine("What is the name? ");
    if (name.isEmpty()) {
      System.out.println("Please, don’t type "
                  + "an empty name next time.");
      return;
    }
    Date birth = Console.readDate("What is the birth date? ");
    if (birth == null) {
      System.out.println("Please, type a "
                  + "valid date next time.");
      return;
    }
    Zoo.INSTANCE.addEmployee(new Employee(name, birth));
    System.out.println("Thank you! A new employee was hired.");
  }
}

private static class FireEmployeeOption extends MenuOption {
  public FireEmployeeOption() {
    super("Fire Employee");
  }

  @Override
  public void run() {
    ElementSelectionMenu menu = new ElementSelectionMenu(
          "Select employee", Zoo.INSTANCE.getEmployees());
    menu.run();
    Employee employee = (Employee) menu.getSelection();
    if (employee != null) {
      Zoo.INSTANCE.removeEmployee(employee);
      System.out.println("You fired "
                         + employee.getName() + ":-(");
    } else {
      System.out.println("I’m glad you fired none!");
    }
  }
}

private static class ListEmployeeOption extends MenuOption {
  public ListEmployeeOption() {
    super("List Employees");
  }

  @Override
  public void run() {
    Employee[] employees = Zoo.INSTANCE.getEmployees();
    if (employees.length == 0) {
      System.out.println("There are no employees in the Zoo. "
                          + "Hire some!");
      return;
    }
    for (Employee employee : employees)
    System.out.println(employee);
  }
}

FinanceMenu() {
  super("Finance department", new BalanceOption(),
          new TicketPriceOption(), new DonateOption() ,
        new HireEmployeeOption(), new FireEmployeeOption(),
        new ListEmployeeOption() );
}
```

Area.java:

```
public Point sw;
public Point ne;
```

```
   > When Bob finishes deleting Employee:

If mode is CI
  CHECK-IN ``Employee management support.''
Else
  PUBLISH
  WAIT for Bob to finish task 7
  CHECK-IN
    ``Added utility function.
    Some changes on animal's energy, and the foods.
    Added birth date.
    Some more changes to animals.
    Menu stuff changed.
    Changes in tests.
    Employee management support.''
<
```

A.2 Study Two

A.2.1 Subject Guide

Subject Guide II

Hello!

The goal of this experiment is to evaluate how conflicts between concurrent tasks are managed by software developers working as a team.

```
> Please, turn off or silence your mobile phone <
```

In the preparation for this experiment you were asked to watch a video in which a programmer used some views in Eclipse to manage conflicts between his tasks and those of a co-worker.

Likewise, during this experiment, you (Bob) and your co-worker (Anne) will modify the Zoo application at the same time, and be watched on how you manage conflicts.

Succinctly, a Zoo has zones (Zone) each one occupying its area (Area). Each zone has animals of a single species (the non-abstract classes, e.g., Crocodile), and the animals get energy by eating (Food). The menus have options for managing the Zoo — e.g., to list zones, allocate animals to zones, and sell tickets.

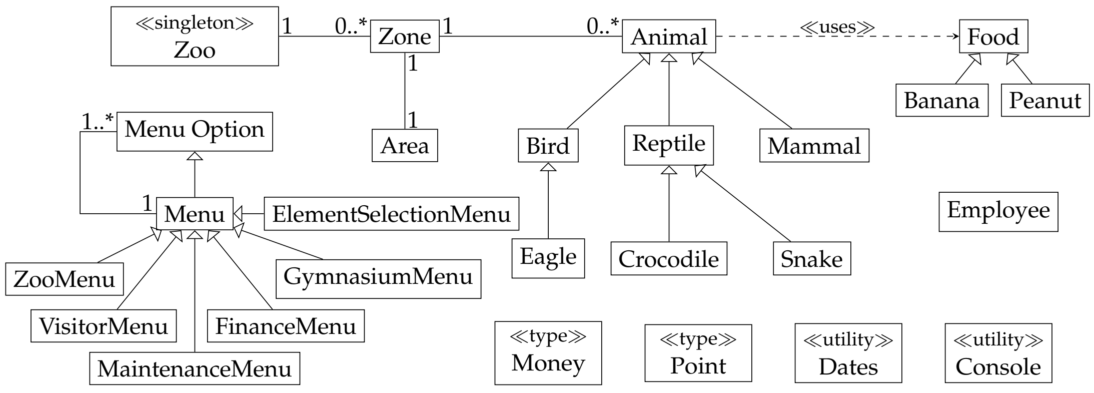

Figure 1: Conceptual class diagram.

A video will be recorded during the experiment with everything you do in the computer. You must say out loud your actions, observations, and the rationale of the decisions you take during your tasks, like when you start or end a task, you see a conflict, you decide some action to resolve a conflict, you see something interesting in a view, etc.

Privacy Statement

The video is for scientific use only, and all public conclusions will be anonymous. Please, do not disclose the contents of this experiment to not affect this research.

During your tasks you must respect the following points:

- Save the files as you change them;

- Fix compilation errors as you find them;
- Resolve the conflicts, thinking on the best resolutions for your team. If you need to negotiate a resolution you may talk to Anne via the chat;
- Do not check in.

Are you ready? This is a good time for clarifications. Please do so if you need!

```
   > Do double-click in ``Start Screen Recording'', and then
do ``CMD + TAB'' to switch to Eclipse <
```

Task List

Do each tasks so that the indicated unit tests are run successfully. (Suggestion: do not modify menus, and begin each task by looking first to its unit tests.)

Task 1) The management wants to be able to move animals to any location inside their zone, but never outside of this. Note that it is not necessary to verify if two animals occupy the same location. Unit tests to pass: AnimalTests.testValidMove() and AnimalTests.testInvalidMove().

Task 2) The zoo will have lions too (daily tax 5.00€), so it is necessary to create this subclass of felines (do it in the animals package where class Feline is). In addition, add to felines the possibility to calculate the tax to pay for them between two dates. Suggestion: some methods in the classes zoo.utils.Dates and zoo.utils.Money may help you doing this task. Unit tests to pass: AnimalTests.testFelineTax().

Task 3) The list of zones must present for each zone its area in square meters (m^2). Unit tests to pass: ZooTests.testAreaM2().

When you have finished all these tasks,

```
   > Click in ``Stop Recording'' in the top bar in your
desktop <
```

A.2.2 Confederate Guide

This guide will provoke the following conflicts with the tasks of the subject in the previous guide:

- Subject adds `Animal.move(...)`, and the confederate adds `Mammal.move(...)` (method override conflict — an indirect conflict);
- Subject adds new concrete subclasses of `Animal`, and the confederate adds the static method `Zoo.getPrice()` (missing method conflict — an indirect conflict);
- Subject and confederate modified different elements in class `Area` (pseudo-direct conflict — a direct conflict).

The commands in the confederate guide, denoted as `> command <`, are meant to publish the code typed by the confederate, after the previous `> command <`, to provoke the above conflicts.

Guide Confederate II

```
  > Tell subjects that they cannot change the visibility of
existing methods.  <
```

Area.java:

```java
final Point sw;
final Point ne;

public int getPerimeter() {
  return 2*((ne.x - sw.x)+(ne.y - sw.y));
}
```

```
  > Before Bob starts any task:  PUBLISH <
```

Zoo.java:

```java
public static Money getPrice(Class<?> species) {
  assert isConcreteAnimal(species);
  try {
    Method getPrice = species.getMethod("getPrice");
    return (Money) getPrice.invoke(null);
  } catch (Exception e) {
    e.printStackTrace();
  }
  return null;
}
```

OtherTests.java:

```java
import junit.framework.TestCase;
import zoo.Zoo;
import zoo.utils.Money;

public class OtherTests extends TestCase {
  private static final Money MONEY_ZERO = new Money("0.00");

  public void testAnimalGetPrice() {
    for (Class<?> species : Zoo.SPECIES) {
      Money price = Zoo.getPrice(species);
      String name = species.getSimpleName();
      assertNotNull("Price not defined for "+ name.toLowerCase()
          + "s. Every concrete animal class must define "
          + "a public static Money getPrice() method.", price);
    assertTrue("The price of "+ name +"s must be higher than "
        + MONEY_ZERO, price.gt(MONEY_ZERO));
   }
  }
}
```

Tiger.java e Crocodile.java:

```
public static Money getPrice() {
  return new Money("100.00");
}
```

```
> When Bob starts task 2:  PUBLISH <
```

Mammal.java (use the same method name chosen by Bob in task 1):

If Bob adds “move distance”, Anne (the confederate) adds “move to” (choose the method below that matches the same argument list added by Bob):

```
public void move(int x, int y) throws ZooException {
  Point p = new Point(x, y);
  if (!getZone().isInside(p))
    throw new ZooException();
  setLocation(p);
}
```

```
public void move(Point p) throws ZooException {
  if (!getZone().isInside(p))
    throw new ZooException();
  setLocation(p);
}
```

But if Bob adds “move to”, Anne (the confederate) adds “move distance” (choose the method below that matches the same argument list added by Bob):

```
public void move(int distancex, int distancey)
    throws ZooException {
  Point loc = getLocation();
  int x = loc.x + distancex;
  int y = loc.y + distancey;
  Point p = new Point(x, y);
  if (!getZone().isInside(p))
    throw new ZooException();
  setLocation(p);
}
```

```java
public void move(Point distance) throws ZooException {
  Point loc = getLocation();
  int x = loc.x + distance.x;
  int y = loc.y + distance.y;
  Point p = new Point(x, y);
  if (!getZone().isInside(p))
    throw new ZooException();
  setLocation(p);
}
```

```
> When Bob starts task 3:  PUBLISH <
```