CSSE 120 – Fundamentals of Software Development I

Exam 2: Fall term 2003-2004

Name:_____ Box #: _____ Time allowed: 120 minutes

Instructor (check one): □Anderson □Azhar □Mutchler

Instructions: This test has 3 parts:

- **Part 1: closed-book.**
- **Part 2: open-book, paper-and-pencil problems.**
- **Part 3: open-book, on-the-computer problems.**

For Part 1 (closed-book):

- **You may NOT use any external resources (no book, notes, computer, etc.)**
- **Turn in Part 1 before you begin using any external resources.**
  - However, you may *read* Parts 2 and 3 (and begin thinking about those parts) before you turn in Part 1.

For Part 2 (open-book, paper-and-pencil) and Part 3 (open-book, on-the-computer):

- You may access your computer and any materials that you brought with you (books, notes, etc), but only after you turn in Part 1.
- You may use the network ONLY to access the CSSE 120 web site (and items to which it links).
  - You must NOT use ANY form of communication. No email, no chat, not anything!
  - **Disable** all chat tools (AIM, ICQ, etc) before the exam starts.

For paper-and-pencil problems (Parts 1 and 2):

- Write all answers on these pages ***legibly***.
  - Use additional sheets as necessary.
  - If you use additional sheets, indicate on the exam where your answer continues.

Regarding code that you write:

- All the code you turn in should be correct, efficient, and use good style.
- However, due to time constraints, no documentation is required on the *paper-and-pencil* parts.
- Appropriate documentation *is required* on the *on-the-computer* part.

Problem	Points available	Your score
1.	5	
2.	5	
3.	10	
4.	15	
5a.	10	
5b.	15	
5c.	15	
7a.	15	
7b.	15	
Total	70	

Part 1 (closed-book):

- **You may NOT use any external resources in Part 1 (no book, notes, computer, etc.)**
- **Turn in Part 1 before you begin using any external resources.**
  - However, you may *read* Parts 2 and 3 (and begin thinking about those parts) before you turn in Part 1.

1. (5 points: 1 point for each item). Consider some of the types of relationships. For example, Paddle **is-a** Wall, Table **has-a** Wall… Classify the relationships below according to the type of relationship, where you should use Rose-Hulman Institute of Technology as the model School. This means that if ClassRoom is generally adjacent to another ClassRoom at Rose-Hulman, then we can consider that to be generally true.

Class 1	**is-a or has-a**	**Class 2**
Student		Person
School		Student
School		ClassRoom
ClassRoom		Wall
Chair		Furniture

2. (5 points: 1 point for each item). We talked about relationships being classified based on multiplicity (1-1, 1-many, many-1). For example, Student-to-Class is a many-to-many, Table-to-Wall is a 1-to-many, Country-CurrentPresident is 1-1. Classify the relationships in the following table according to their type, where you should use Rose-Hulman Institute of Technology as the model School. This means that if Rose-Hulman has one CurrentPresident, then we can consider that to be generally true.

Class 1	**Indicate: 1-1 or 1-many or many-1 or many-many**	**Class 2**
Teacher		Class
Student		SocialSecurityNumber
Student		Teacher
School		Chapel
Department (e.g., CSSE)		School

3. (10 points, 1 point each) What do the following statements do?

a. `public class Paddle extends Wall implements Runnable, Bounceable`

b. `new Thread(this).start();`

c. `super();`

d. `this(myName, DEFAULT_WEIGHT, DEFAULT_WEIGHT);`

e. `super.paintComponent(g);`

f. `this.move();`

g. `this.myBall.move();`

h. `Square myBoard[][] = new Square[X][Y];`

i. The following code segment

```
Square(int i, int j) {
   this.xPosition = i;
   this.yPosition = j;
}
```

j. The following code segment

```
for (i = 0; i < X; i++) {
   for (j = 0; j < Y; j++) {
         myBoard[i][j] = new Square(i,j);
   }
}
```

Part 2 (open-book, paper-and-pencil):

- **Turn in Part 1 before you begin using any external resources (book, notes, computer, etc) on Part 2.**
- Note the rules for Part 2, as stated on the cover page of this exam.

4. (15 points). Write a method called **`int sum5Even(int n)`** such that the method takes as input an positive integer **`n`**, and returns the sum of all positive numbers strictly less than **`n`** that are divisible by 2 or 5. For example:

THESE ARE ONLY EXAMPLES OF TEST DATA. YOUR PROGRAM MUST WORK FOR ANY POSITIVE INTEGER n.		
The call	returns	Because
`sum5Even(3)`	**`2`**	2 is the only number added
`sum5Even(5)`	**`6`**	that is the sum of 2 and 4
`sum5Even(10)`	**`25`**	that is the sum of 2, 4, 5, 6, and 8
`sum5Even(11)`	**`35`**	that is the sum of 2, 4, 5, 6, 8, and 10 (Note 10 is must not be double counted)

(Hints: A number is divisible by 2, if it leaves a remainder of 0 when divided by 2. Also, you may find the "%" operator useful here.)

5. (35 points). **Read the entire question before working on individual parts.** Consider the game of Chess. Today, you will design and present a Class Diagram for Chess.

   Be mindful of the following requirements:

   i. This problem has several parts and each part will lead you to the next. So, do them in the prescribed order.
   ii. This question is about **Player-vs-Player** game and you must **not** address the user interface component(s). (We just want you to focus on modeling the game of Chess itself).
   iii. You can find the description of Yahoo! Chess at the following URLs:

   http://games.yahoo.com/games/rules/chess/history.html?page=ch

   http://games.yahoo.com/games/rules/chess/basics.html?page=ch

   http://games.yahoo.com/games/rules/chess/pieces.html?page=ch (and associated links for Queen, Knight, Bishop, Rook, and Pawn).

   iv. You must not address any implementation details (such as the actual code that is needed) or other rules of chess (such as check or castling).

   a. (10 points). Your first step is to identify the Classes in the above problem. Be mindful of the following requirements:
      i. You must have at least 9 classes.
      ii. You do need to have relationships, fields or methods in your list yet.
      iii. Be prepared to change this list after you have done the other parts.

      On an extra sheet of paper, go ahead and list the classes.

   b. On an extra sheet of paper, sketch a class diagram showing the relationships in Part a.

   . Be mindful of the following requirements:
      i. See http://www.rose-hulman.edu/class/csse/csse120/resources/classNotes/session20/PingPongDesign.doc Page 1 for example.
      ii. You must have at two or more "is-a" relationships and one or more "has-a" relationships. If you don't, go back to Part a, and modify your list of classes.
      iii. You should not have fields or methods in your diagram yet.
      iv. Where necessary show the multiplicity of the relationships.
      v. You need not worry about interfaces for the purpose of this class diagram

   c. On an extra sheet of paper, sketch the detailed object diagrams for three of the classes in Part b on an extra sheet of paper.

      i. (See http://www.rose-hulman.edu/class/csse/csse120/resources/classNotes/session20/PingPongDesign.doc Page 3 for example).

      ii. You must pick the following classes:

      (1) The class King;

      (2) One Class that acts as superclass of another class (just the superclass not the subclass).

      (3) One class that "has-a" another class (again just the class that contains, not the class that is contained).

Part 3 (open-book, on-your-computer):

- **Turn in Part 1 before you begin using any external resources (book, notes, computer, etc) on Part 3.** Note the rules for Part 3, as stated on the cover page.

6. (35 points). **Do this problem on your computer.** To do so:

   Step 1: Go to the CSSE 120 ***home page*** and download and unzip the *CatAndMouse* project.

   Step 2: Examine the CatAndMouse class diagram, on a separate handout, as follows:

   - Look closely at page 1 (the problem statement).
   - Look closely at page 2 (the class diagram that shows only the classes and their relationships). *We* implemented *all* the classes on the left-hand-side of that page. *You* will implement ONLY the following classes (all on the right-hand-side of the class diagram): **Animal Cat Mouse CatsAndMice**
   - Briefly skim page 3 – we implemented all the classes on page 3. You should NOT change any of those classes. You can succeed at this project without looking at the code for any of those classes.
   - Look closely at page 4, which shows the class diagram for the CatsAndMice class. We implemented all of this class *except* its constructor.
   - Look closely at page 5, which shows the class diagram for the Cat, Mouse and Animal classes. You will implement all of these classes.

   Step 3: Implement the CatsAndMice, Cat, Mouse and Animal classes per the class diagram and the problem statement, **per the following iterative enhancement plan:**

   - Step 3a: Implement empty Animal, Cat and Mouse classes (no fields, no methods).
     - The project *will have compile errors in OUR parts of the code* until you have implemented these skeleton Animal, Cat and Mouse classes.
   - Step 3b: Implement ALL the methods required by the class diagrams for Animal, Cat and Mouse, but with **empty bodies** (or trivial bodies for those methods that require a returned value).
   - Step 3c: Get a single Mouse to work properly. Some hints about what you might work on first:
     - Implement the Animal's *getRoom* method, since that method is how our classes communicate with your classes.
     - Implement the CatsAndMice constructor.
     - Implement the relevant Animal methods, then the relevant Mouse methods.
     - **Compile and test frequently!**
   - Step 3d: Get the other two mice to work properly.
   - Step 3e: Get the cats to work properly.

- Step 4*:* When done, copy your entire *CatAndMouse* project to your *turnin* folder under **Exam2**
- **Important:** If you don't understand the problem statement, ask your instructor or an assistant.
- **Important:** If you have any compile-time errors that you cannot resolve, you may *twice* ask for help from your instructor or an assistant. After that, you are on your own.

Problem statement:

Two cats and three mice are in a house with 10 rooms.

- The cats and mice move from room to room randomly, as long as they are alive.
- When a cat enters a room:
  - If a mouse is in that room, the cat eats the mouse.
  - Then the cat sleeps in that room for a while.
- When a mouse enters a room:
  - The mouse sometimes leaves immediately,
  - but otherwise it sleeps in that room for a while.

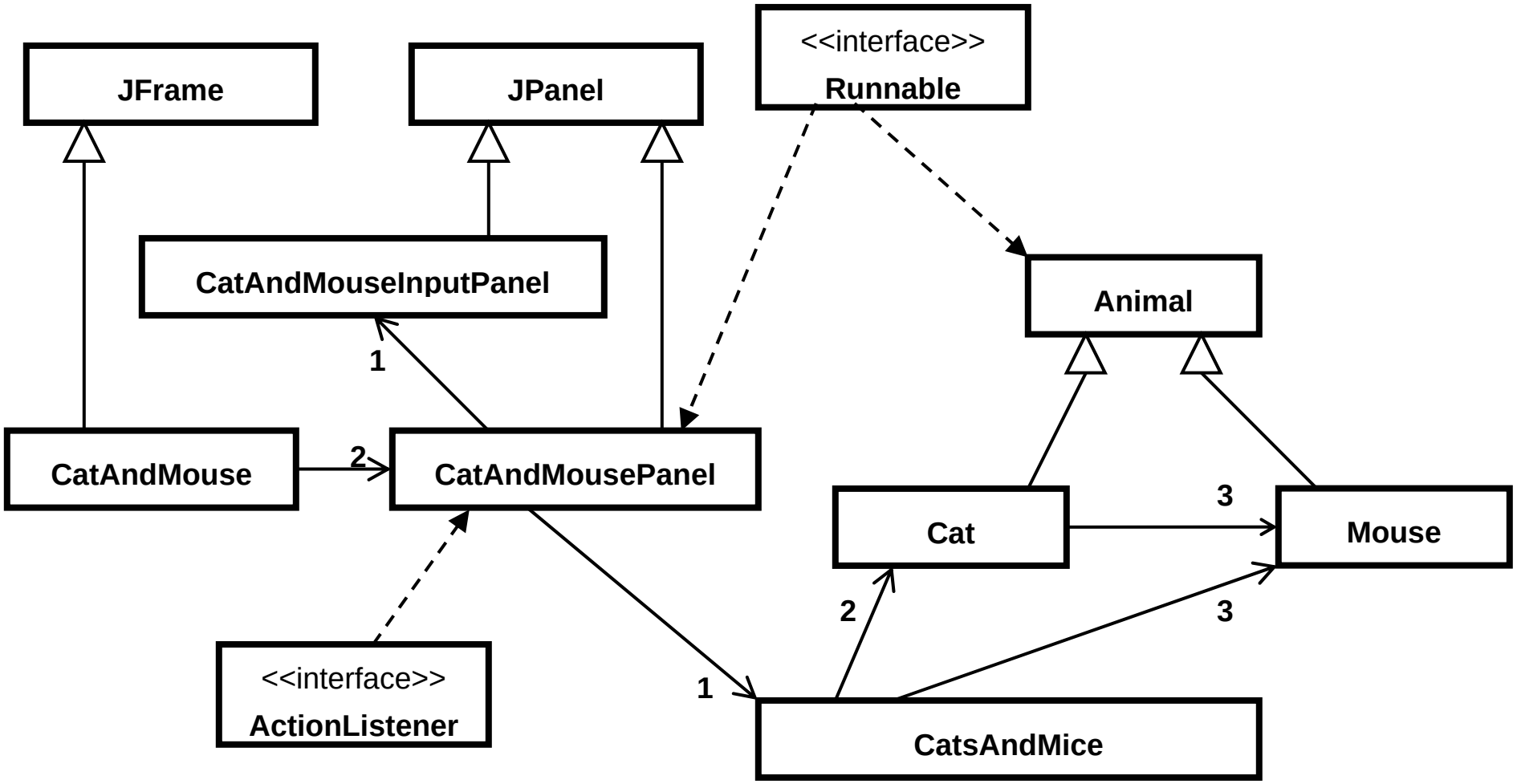
Notes:

1. The rooms are numbered from 1 to 10.
2. If there is more than one mouse in the room, the cat eats all of them.
3. Each animal has a “timeToSleep” that is the number of milliseconds that the animal sleeps in a room before moving to another randomly chosen room.
4. Each mouse has a “roomToLeaveImmediately” that is a number between 1 and 10. When a mouse enters a room, it leaves the room immediately if the room it enters is that mouse’s roomToLeaveImmediately. For example, if a mouse’s roomToLeaveImmediately is 4, then that mouse immediately leaves room 4 whenever it enters it, but sleeps for a while whenever it enters any other room.
5. When an animal moves to a new (randomly chosen) room, the new room should really be new (i.e., not the same room in which the animal currently is in).

Problem statement (repeated for your convenience):

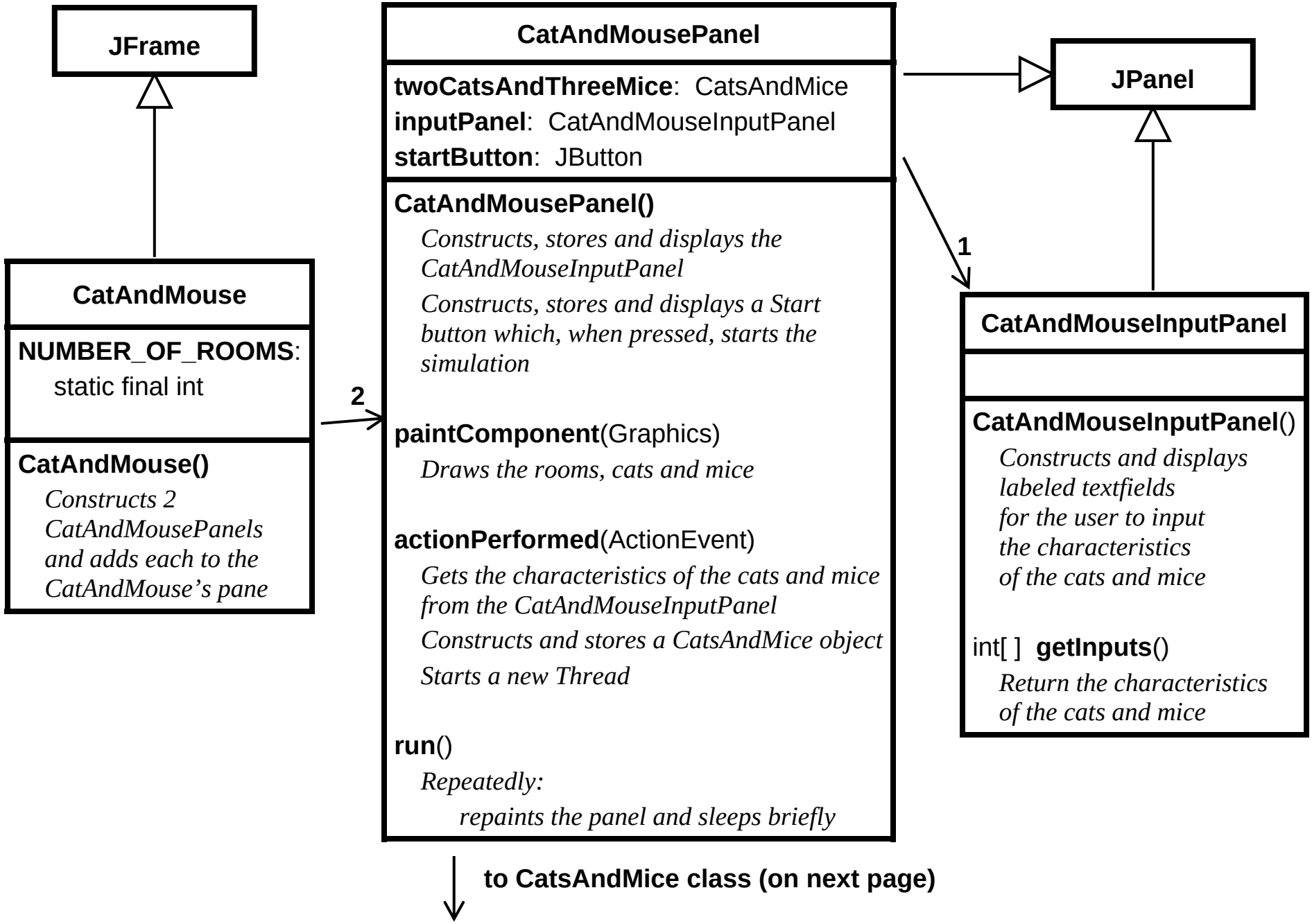
Two cats and three mice are in a house with 10 rooms. The cats and mice move from room to room randomly, as long as they are alive. When a cat enters a room, if a mouse is in that room, the cat eats the mouse. Then the cat sleeps in that room for a while. When a mouse enters a room, the mouse sometimes leaves immediately, but otherwise it sleeps in that room for a while.

Classes and relationships



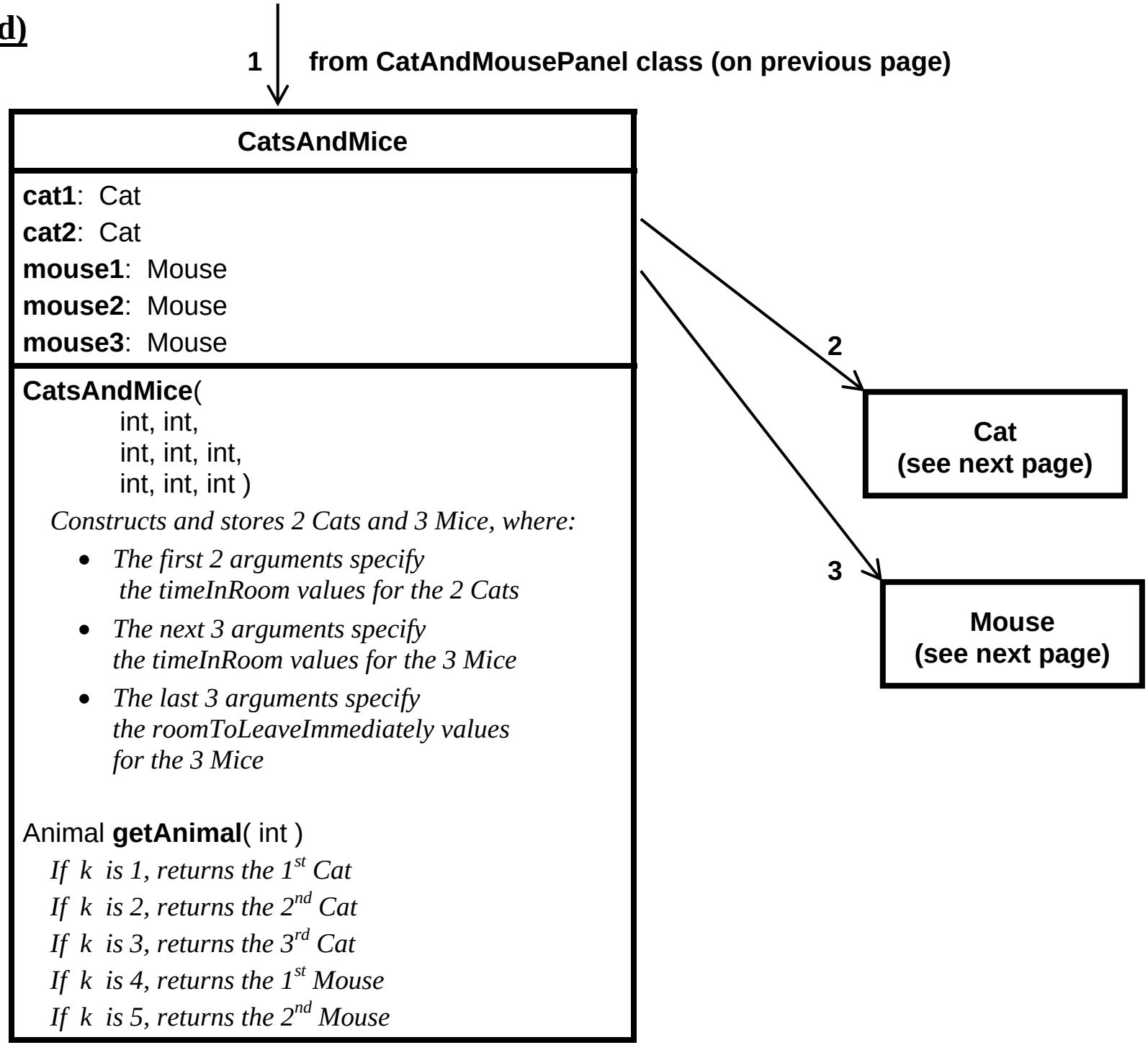
Classes:

We have implemented *all* the classes on this page



Classes (continued)

You implement only the *constructor* of this class



Classes (continued) *You* implement *all* of these classes

Animal

isAlive: boolean
True if the animal is alive, else false
room: int
The room that the animal is currently in
timeInRoom: int
Number of milliseconds that the animal sleeps in a room

Animal()
Set the timeInRoom for this Animal to1000 milliseconds.
Start a new Thread
Animal(int)
Set the timeInRoom for this Animal to the given argument
Start a new Thread

enterRoom(int)
Simulate the Animal entering the given room by:
Sleep in the room for this Animal's timeInRoom

int **getRoom**()
Return the room that the animal is currently in

boolean **isAlive**()
Return true if the Animal is alive, else return false

run()
Repeat:
1. Choose a random room (i.e., a random number between 1 and CatAndMouse.NUMBER_OF_ROOMS)
2. Enter that room
until the Animal is no longer alive

Cat

mouse1: Mouse
mouse2: Mouse
mouse3: Mouse

Cat(Mouse, Mouse, Mouse, int)
Set the Cat's Mice to the 1st three arguments
Set the Cat's timeInRoom to the 4th argument

enterRoom(int)
Simulate the Cat entering the given room by:
1. For each of the Cat's 3 Mice:
If the Mouse is in the same room that the cat entered, the Cat eats the Mouse (i.e., the Mouse is eaten)
2. Sleep in the room for this Cat's timeInRoom

Mouse

roomToLeaveImmediately: int

Mouse(int, int)
Set the Mouse's timeInRoom to the 1st argument
Set the Mouse's roomToLeaveImmediately to the 2nd argument

beEaten()
Make the Mouse be dead (i.e., not alive)

enterRoom(int)
Simulate the Mouse entering the given room by:
- *If the given room is the Mouse's roomToLeaveImmediately, the Mouse doesn't sleep (thereby leaving the room immediately)*
- *Otherwise, the Mouse sleeps in the room for this Mouse's timeInRoom*