**Университет ИТМО**

**Факультет программной инженерии и компьютерных технологий**

**Лабораторная работа №4 по Программированию**

**Выполнил: Богатов Александр Сергеевич**

**Группа: P3133**

**Вариант: 33011**

**Преподаватель: Горбунов Михаил Витальевич**

**Санкт-Петербург**

**2020**

**Задание:**

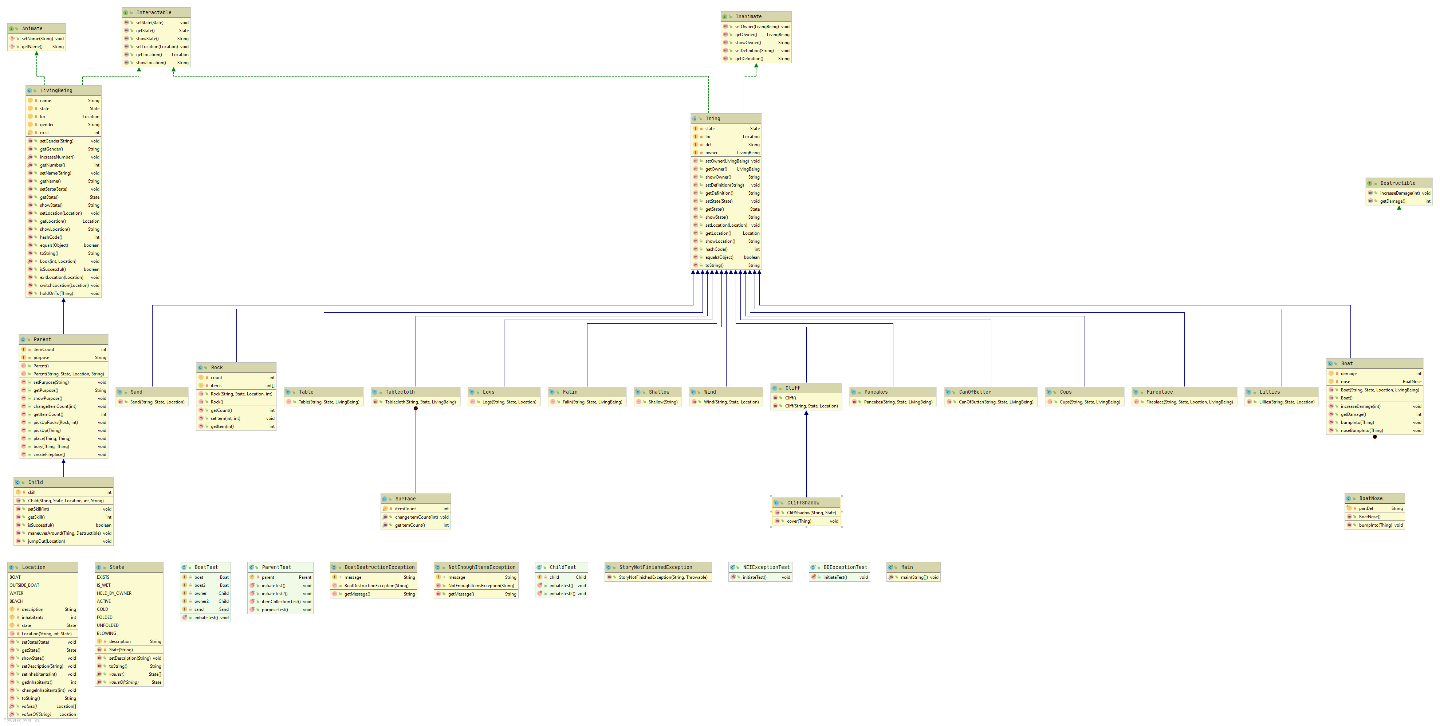
Описание предметной области, по которой должна быть построена объектная модель:

*Все потянулись через борт посмотреть. Изящно лавируя между отмелями, он провел лодку к берегу. Ее нос мягко ткнулся в песок. Муми-тролль спрыгнул с фалинем (так называется трос, которым лодка крепится к причалу) на сушу, и вскоре берег ожил, наполнился суетой и бурной деятельностью. Муми-мама сложила из камней очаг, чтобы подогреть оладьи, натаскала дров, расстелила на песке скатерть и на каждый угол положила по камню, чтобы ее не унесло ветром. Затем расставила чашки, закопала банку с маслом в сырой песок в тени от скалы и под конец украсила стол букетом береговых лилий.*

Программа должна удовлетворять следующим требованиям:

1. В программе должны быть реализованы 2 собственных класса исключений (checked и unchecked), а также обработка исключений этих классов.
2. В программу необходимо добавить использование локальных, анонимных и вложенных классов (static и non-static).

**Диаграмма классов объектной модели:**



**Исходный код программы:**

Main.java

1. **package** moumie;
3. **public** **class** Main {
4. **public** **static** **void** main(String[] args) **throws** BoatDestructionException, NotEnoughItemsException {
5. Wind wind = **new** Wind("Ветер", State.BLOWING, Location.BEACH);
6. Parent mom = **new** Parent("Муми-мама", State.EXISTS, Location.BOAT, "Ж");
7. Child son = **new** Child("Муми-тролль", State.EXISTS, Location.BOAT, 10, "М");
8. Shallow sh1 = **new** Shallow("Отмель");
9. Shallow sh2 = **new** Shallow("Отмель");
10. Shallow sh3 = **new** Shallow("Отмель");
11. Sand sand = **new** Sand("Песок", State.IS\_WET, Location.BEACH);
12. Boat boat = **new** Boat("Лодка", State.EXISTS, Location.WATER, son);
13. Falin falin = **new** Falin("Фалинь", State.EXISTS, son);
14. Rock rocks = **new** Rock("Камень", State.EXISTS, Location.BEACH, 10);
15. Pancakes food = **new** Pancakes("Оладьи", State.COLD, mom);
16. Logs logs = **new** Logs("Дрова", State.EXISTS, Location.BEACH);
17. Tablecloth cloth = **new** Tablecloth("Скатерть", State.FOLDED, mom);
18. Cups cups = **new** Cups("Чашки", State.EXISTS, mom);
19. CanOfButter can = **new** CanOfButter("Банка с маслом", State.EXISTS, mom);
20. Cliff cl = **new** Cliff("Скала", State.EXISTS, Location.BEACH);
21. CliffShadow shadow = **new** CliffShadow("Тень скалы", State.EXISTS);
22. shadow.cover(sand);
23. Lillies lillies = **new** Lillies("Лилии", State.EXISTS, Location.BEACH);
24. Table table = **new** Table("Стол", State.EXISTS, mom);
25. LivingBeing.Look(2, Location.OUTSIDE\_BOAT);
26. **try** {
27. son.maneuverAround(sh1, boat);
28. boat.noseBumpInto(sand);
29. son.holdOnTo(falin);
30. son.jumpOut(Location.BEACH);
31. mom.switchLocation(Location.BEACH);
32. **if**(Location.BEACH.getInhabitants()>=2) {
33. Location.BEACH.setState(State.ACTIVE);
34. mom.setState(State.HURRY);
35. son.setState(State.HURRY);
36. System.out.println("Обитатели берега суетятся");
37. Location.BEACH.showState();
38. }
39. mom.pickUpRocks(rocks, 12);
40. mom.createFireplace();
41. mom.pickUp(logs);
42. mom.place(cloth, sand);
43. **if**(mom.getItemCount() >= 4) {
44. **int** i = mom.getItemCount();
45. **while**(Tablecloth.Surface.getItemCount() < 4) {
46. mom.place(rocks, cloth);
47. }
48. }
49. **else** {
50. mom.pickUpRocks(rocks, 4);
51. **if**(mom.getItemCount() < 4) {
52. **throw** **new** NotEnoughItemsException("Недостаточно предметов");
53. }
54. **int** i = mom.getItemCount();
55. **while**(mom.getItemCount() > i - 4) {
56. mom.place(rocks, cloth);
57. }
58. }
59. mom.place(cups, cloth);
60. mom.bury(can, sand);
61. mom.pickUp(lillies);
62. mom.place(lillies, table);
63. } **catch**(BoatDestructionException|NotEnoughItemsException e) {
65. **throw** **new** StoryNotFinishedException("История не может быть завершена", e);
66. }
67. }
68. }

Animate.java

1. **package** moumie;
2. **public** **interface** Animate {
3. **void** setName(String name);
4. String getName();
5. }

Inanimate.java

1. **package** moumie;
2. **public** **interface** Inanimate {
3. **void** setOwner(LivingBeing owner);
4. LivingBeing getOwner();
5. String showOwner();
6. **void** setDefinition(String def);
7. String getDefinition();
8. }

Interactable.java

1. **package** moumie;
2. **public** **interface** Interactable {
3. **void** setState(State state);
4. State getState();
5. String showState();
6. **void** setLocation(Location loc);
7. Location getLocation();
8. String showLocation();
9. }

Destructible.java

1. **package** moumie;
3. **public** **interface** Destructible {
4. **public** **void** increaseDamage(**int** i);
5. **public** **int** getDamage();
6. }

LivingBeing.java

1. **package** moumie;
2. **import** java.util.Objects;
4. **public** **abstract** **class** LivingBeing **implements** Interactable, Animate {
5. **private** String name;
6. **private** State state;
7. **private** Location loc = **null**;
8. **private** String gender;
9. **private** **static** **int** exist;
10. **public** **void** setGender(String g) {
11. **this**.gender = g;
12. }
13. **public** String getGender() {
14. **return** **this**.gender;
15. }
16. **public** **static** **void** increaseNumber() {
17. LivingBeing.exist = LivingBeing.exist+1;
18. }
19. **public** **static** **int** getNumber() {
20. **return** LivingBeing.exist;
21. }
22. @Override
23. **public** **void** setName(String name) {
24. **this**.name = name;
25. }
26. @Override
27. **public** String getName() {
28. **return** **this**.name;
29. }
30. @Override
31. **public** **void** setState(State state) {
32. **this**.state = state;
33. }
34. @Override
35. **public** State getState() {
36. **return** **this**.state;
37. }
38. @Override
39. **public** String showState() {
40. **return** **this**.state.toString();
41. }
42. @Override
43. **public** **void** setLocation(Location loc) {
44. **if**(**this**.loc != **null**) {
45. **this**.loc.changeInhabitants(-1);
46. }
47. **this**.loc = loc;
48. **this**.loc.changeInhabitants(1);
50. }
51. @Override
52. **public** Location getLocation() {
53. **return** **this**.loc;
54. }
55. @Override
56. **public** String showLocation() {
57. **return** **this**.loc.toString();
58. }
59. @Override
60. **public** **int** hashCode() {
61. **return** Objects.hash(name, state, loc);
62. }
63. @Override
64. **public** **boolean** equals(Object obj) {
65. **if**(obj == **null** || obj.getClass() != **this**.getClass())
66. **return** **false**;
67. **if**(**this** == obj)
68. **return** **true**;
69. LivingBeing other = (LivingBeing) obj;
70. **return** Objects.equals(name, other.name);
71. }
72. @Override
73. **public** String toString() {
74. **return** ("Живое существо по имени " + getName());
75. }
76. **static** **void** Look(**int** i, Location loc) {
77. **if**(i<LivingBeing.getNumber())
78. System.out.println("Некоторые посмотрели " + loc.toString());
79. **if**(i==LivingBeing.getNumber()) {
80. System.out.println("Все посмотрели " + loc.toString());
81. }
82. }
83. **public** **boolean** isSuccessful() {
84. **return** **true**;
85. }
86. **public** **void** exitLocation(Location loc) {
87. **this**.loc.changeInhabitants(-1);
88. setLocation(loc);
89. **this**.loc.changeInhabitants(1);
90. }
91. **public** **void** switchLocation(Location loc) {
92. System.out.println(getName() + " перешел из " + **this**.showLocation() + " в " + loc.toString());
93. exitLocation(loc);
94. }
95. **public** **void** holdOnTo(Thing thing) {
96. **if**(thing.getOwner().equals(**this**)) {
97. System.out.println(getName() + " держит " + thing.getDefinition());
98. thing.setState(State.HELD\_BY\_OWNER);
99. }
100. }
101. }

Thing.java

1. **package** moumie;
2. **import** java.util.Objects;
4. **public** **abstract** **class** Thing **implements** Interactable, Inanimate{
5. **private** State state;
6. **private** Location loc;
7. **private** String def;
8. **private** LivingBeing owner;
9. @Override
10. **public** **void** setOwner(LivingBeing owner) {
11. **this**.owner = owner;
12. }
13. @Override
14. **public** LivingBeing getOwner() {
15. **return** **this**.owner;
16. }
17. @Override
18. **public** String showOwner() {
19. **return** **this**.owner.toString();
20. }
21. @Override
22. **public** **void** setDefinition(String def) {
23. **this**.def = def;
24. }
25. @Override
26. **public** String getDefinition() {
27. **return** **this**.def;
28. }
29. @Override
30. **public** **void** setState(State state) {
31. **this**.state = state;
32. }
33. @Override
34. **public** State getState() {
35. **return** **this**.state;
36. }
37. @Override
38. **public** String showState() {
39. **return** **this**.state.toString();
40. }
41. @Override
42. **public** **void** setLocation(Location loc) {
43. **this**.loc = loc;
44. }
45. @Override
46. **public** Location getLocation() {
47. **return** **this**.loc;
48. }
49. @Override
50. **public** String showLocation() {
51. **return** **this**.loc.toString();
52. }
53. @Override
54. **public** **int** hashCode() {
55. **return** Objects.hash(def, state, loc);
56. }
57. @Override
58. **public** **boolean** equals(Object obj) {
59. **if**(obj == **null** || obj.getClass() != **this**.getClass())
60. **return** **false**;
61. **if**(**this** == obj)
62. **return** **true**;
63. Thing other = (Thing) obj;
64. **return** Objects.equals(def, other.def);
65. }
66. @Override
67. **public** String toString() {
68. **return** getDefinition();
69. }
70. }

Boat.java

1. **package** moumie;
3. **public** **final** **class** Boat **extends** Thing **implements** Destructible {
4. **private** **int** damage = 0;
5. **private** BoatNose nose;
6. **public** **void** increaseDamage(**int** i) {
7. **this**.damage = **this**.damage + i;
8. }
9. **public** **int** getDamage() {
10. **return** **this**.damage;
11. }
12. **public** Boat(String def, State state, Location loc, LivingBeing owner) {
13. setDefinition(def);
14. setState(state);
15. setLocation(loc);
16. setOwner(owner);
17. nose = **new** BoatNose();
18. }
19. **public** Boat() {
21. }
22. **public** **class** BoatNose {
23. **private** **final** String partDef = "Нос лодки";
24. **public** BoatNose() {
26. }
27. **public** **void** bumpInto(Thing thing) **throws** BoatDestructionException {
28. System.out.print("Носом ");
29. Boat.**this**.bumpInto(thing);
30. }
31. }
32. **public** **void** bumpInto(Thing thing) **throws** BoatDestructionException {
33. **if**(thing.toString().equals("Песок")) {
34. **if**(getOwner().isSuccessful()) {
35. System.out.println("Лодка мягко уткнулась в " + thing.toString());
36. }
37. **else** {
38. System.out.println("Лодка врезалась в " + thing.toString());
39. increaseDamage(2);
40. **if**(getDamage()>=10) {
41. **throw** **new** BoatDestructionException("Лодка разбилась");
42. }
43. }
44. }
45. }
46. **public** **void** noseBumpInto(Thing thing) **throws** BoatDestructionException {
47. BoatNose nose = **new** BoatNose();
48. nose.bumpInto(thing);
49. }
50. }

CanOfButter.java

1. **package** moumie;
3. **public** **final** **class** CanOfButter **extends** Thing {
4. **public** CanOfButter(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. }

Child.java

1. **package** moumie;
2. **public** **final** **class** Child **extends** Parent {
3. **private** **int** skill;
4. **public** **void** setSkill(**int** i) {
5. **this**.skill = **this**.skill + i;
6. }
7. **public** **int** getSkill() {
8. **return** **this**.skill;
9. }
10. **public** Child(String name, State state, Location loc, **int** i, String g) {
11. setName(name);
12. setState(state);
13. setLocation(loc);
14. setSkill(i);
15. setGender(g);
16. **super**.increaseNumber();
17. }
18. **public** **boolean** isSuccessful() {
19. **if**(getSkill()>=10) {
20. **return** **true**;
21. }
22. **else** **return** **false**;
23. }
24. **public** **void** maneuverAround(Thing thing, Destructible destrcuctible) **throws** BoatDestructionException {
25. **if**(thing.toString().equals("Отмель")) {
26. **if**(isSuccessful()) {
27. **if**(getGender().equals("М"))
28. System.out.println(getName() + " изящно обошел " + thing.toString());
29. **else**
30. System.out.println(getName() + " изящно обошла " + thing.toString());
31. }
32. **else** {
33. **if**(getGender().equals("М"))
34. System.out.println(getName() + " не сумел обойти " + thing.toString());
35. **else**
36. System.out.println(getName() + " не сумела обойти " + thing.toString());
37. destrcuctible.increaseDamage(10);
38. **if**(destrcuctible.getDamage()>=10) {
39. **throw** **new** BoatDestructionException("Лодка разбилась");
40. }
41. }
42. }
43. }
44. **public** **void** jumpOut(Location loc) {
45. **if**(getGender().equals("М"))
46. System.out.println(getName() + " выпрыгнул из " + **this**.showLocation() + " на " + loc.toString());
47. **else**
48. System.out.println(getName() + " выпрыгнула из " + **this**.showLocation() + " на " + loc.toString());
49. exitLocation(loc);
50. }
51. }

Cliff.java

1. **package** moumie;
3. **public** **class** Cliff **extends** Thing {
4. **public** Cliff() {
6. }
7. **public** Cliff(String def, State state, Location loc) {
8. setDefinition(def);
9. setState(state);
10. setLocation(loc);
11. }
12. }

CliffShadow.java

1. **package** moumie;
3. **public** **final** **class** CliffShadow **extends** Cliff {
4. **public** CliffShadow(String def, State state) {
5. setDefinition(def);
6. setState(state);
7. setLocation(**super**.getLocation());
8. }
9. **public** **void** cover(Thing thing) {
10. System.out.println(getDefinition() + " покрывает " + thing.getDefinition());
11. }
12. }

Cups.java

1. **package** moumie;
3. **public** **final** **class** Cups **extends** Thing {
4. **public** Cups(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. }

Falin.java

1. **package** moumie;
3. **public** **final** **class** Falin **extends** Thing {
4. **public** Falin(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. }

Fireplace.java

1. **package** moumie;
3. **public** **final** **class** Fireplace **extends** Thing {
4. **public** Fireplace(String def, State state, Location loc, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setLocation(loc);
8. setOwner(owner);
9. }
10. }

Lillies.java

1. **package** moumie;
3. **public** **final** **class** Lillies **extends** Thing {
4. **public** Lillies(String def, State state, Location loc) {
5. setDefinition(def);
6. setState(state);
7. setLocation(loc);
8. }
9. }

Logs.java

1. **package** moumie;
3. **public** **final** **class** Logs **extends** Thing {
4. **public** Logs(String def, State state, Location loc) {
5. setDefinition(def);
6. setState(state);
7. setLocation(loc);
8. }
9. }

Pancakes.java

1. **package** moumie;
3. **public** **final** **class** Pancakes **extends** Thing {
4. **public** Pancakes(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. }

Parent.java

1. **package** moumie;
2. **public** **class** Parent **extends** LivingBeing {
3. **private** **int** itemCount;
4. **private** String purpose;
5. **public** **void** setPurpose(String p) {
6. **this**.purpose = p;
7. }
8. **public** String getPurpose() {
9. **return** **this**.purpose;
10. }
11. **public** **void** showPurpose() {
12. System.out.println(**this**.getName() + " хочет " + **this**.purpose);
13. }
14. **public** **void** changeItemCount(**int** i) {
15. **this**.itemCount = **this**.itemCount + i;
16. }
17. **public** **int** getItemCount() {
18. **return** **this**.itemCount;
19. }
20. **public** Parent() {
22. }
23. **public** Parent(String name, State state, Location loc, String g) {
24. setName(name);
25. setState(state);
26. setLocation(loc);
27. setGender(g);
28. **super**.increaseNumber();
29. }
30. **public** **void** pickUpRocks(Rock thing, **int** number) {
31. **if**(getItemCount() == 0)
32. System.out.println(getName() + " подбирает несколько " + thing.getDefinition());
33. **if**(thing.toString().equals("Камень")) {
34. **int** i;
35. **int** j = getItemCount();
36. **for**(i = 0; i < thing.getCount(); i++) {
37. **if**(thing.getItem(i) == 1) {
38. thing.setItem(i, 0);
39. changeItemCount(1);
40. }
41. **if**(getItemCount() == j + number) {
42. **break**;
43. }
44. }
45. }
46. thing.setOwner(**this**);
47. }
48. **public** **void** pickUp(Thing thing) {
49. thing.setOwner(**this**);
50. System.out.println(getName() + " подбирает " + thing.getDefinition());
51. }
52. **public** **void** place(Thing thing, Thing spot) {
53. **if**(getGender().equals("Ж"))
54. System.out.println(getName() + " поместила " + thing.getDefinition() + " на " + spot.getDefinition());
55. **else**
56. System.out.println(getName() + " поместил " + thing.getDefinition() + " на " + spot.getDefinition());
57. thing.setLocation(spot.getLocation());
58. **if**(thing.toString().equals("Камень")) {
59. changeItemCount(-1);
60. }
61. **if**(thing.toString().equals("Скатерть")) {
62. thing.setState(State.UNFOLDED);
63. System.out.println(thing.toString() + " " + thing.showState());
64. }
65. **if**(spot.toString().equals("Скатерть")) {
66. Tablecloth.Surface.changeItemCount(1);
67. }
68. }
69. **public** **void** bury(Thing thing, Thing spot) {
70. **if**(getGender().equals("Ж"))
71. System.out.println(getName() + " закопала " + thing.getDefinition() + " в " + spot.getDefinition());
72. **else**
73. System.out.println(getName() + " закопал " + thing.getDefinition() + " в " + spot.getDefinition());
74. thing.setLocation(spot.getLocation());
75. }
76. **public** **void** createFireplace() **throws** NotEnoughItemsException {
77. **if**(getItemCount() < 6) {
78. **throw** **new** NotEnoughItemsException("Недостаточно предметов");
79. }
80. **else** {
81. changeItemCount(-6);
82. **if**(getItemCount() < 0) {
83. //System.out.println("Недостаточно камней");
84. changeItemCount(6);
85. **throw** **new** NotEnoughItemsException("Недостаточно камней");
86. }
87. **else** {
88. Fireplace fireplace = **new** Fireplace("очаг", State.EXISTS, **this**.getLocation(), **this**);
89. **if**(getGender().equals("Ж"))
90. System.out.println(getName() + " сложила " + fireplace.getDefinition());
91. **else**
92. System.out.println(getName() + " сложил " + fireplace.getDefinition());
93. setPurpose("подогреть оладьи");
94. showPurpose();
95. }
96. }
97. }
99. }

Rock.java

1. **package** moumie;
3. **public** **class** Rock **extends** Thing {
4. **private** **int** count;
5. **private** **int**[] items = **new** **int**[count];
6. **public** **int** getCount() {
7. **return** **this**.count;
8. }
9. **public** **void** setItem(**int** i, **int** j) {
10. **this**.items[i] = j;
11. }
12. **public** **int** getItem(**int** i) {
13. **return** **this**.items[i];
14. }
15. **public** Rock(String def, State state, Location loc, **int** count) {
16. setDefinition(def);
17. setState(state);
18. setLocation(loc);
19. **this**.items = **new** **int**[count];
20. **for** (**this**.count = 0; **this**.count < count; **this**.count++) {
21. **this**.items[**this**.count] = 1;
22. }
23. }
24. **public** Rock() {
26. }
27. }

Sand.java

1. **package** moumie;
3. **public** **final** **class** Sand **extends** Thing {
4. **public** Sand(String def, State state, Location loc) {
5. setDefinition(def);
6. setState(state);
7. setLocation(loc);
8. }
9. }

Shallow.java

1. **package** moumie;
3. **public** **final** **class** Shallow **extends** Thing{
4. **public** Shallow(String def) {
5. setDefinition(def);
6. }
7. }

Table.java

1. **package** moumie;
3. **public** **final** **class** Table **extends** Thing {
4. **public** Table(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. }

Tablecloth.java

1. **package** moumie;
3. **public** **final** **class** Tablecloth **extends** Thing {
4. **public** Tablecloth(String def, State state, LivingBeing owner) {
5. setDefinition(def);
6. setState(state);
7. setOwner(owner);
8. setLocation(owner.getLocation());
9. }
10. **public** **static** **class** Surface {
11. **private** **static** **int** itemCount = 0;
12. **public** **static** **void** changeItemCount(**int** i) {
13. itemCount = itemCount + i;
14. }
15. **public** **static** **int** getItemCount() {
16. **return** itemCount;
17. }
18. }
19. }

Wind.java

1. **package** moumie;
3. **public** **final** **class** Wind **extends** Thing {
4. **public** Wind(String def, State state, Location loc) {
5. setDefinition(def);
6. setState(state);
7. setLocation(loc);
8. }
9. }

Location.java

1. **package** moumie;
2. **public** **enum** Location {
3. BOAT("Лодка", 0, State.EXISTS),
4. OUTSIDE\_BOAT("За борт лодки", 0, State.EXISTS),
5. WATER("Вода", 0, State.EXISTS),
6. BEACH("Берег", 0, State.EXISTS);
8. **private** String description;
9. **private** **int** inhabitants;
10. **private** State state;
12. **public** **void** setState(State state) {
13. **this**.state = state;
14. }
15. **public** State getState() {
16. **return** **this**.state;
17. }
18. **public** **void** showState() {
19. System.out.println(toString() + " " + **this**.getState().toString());
20. }
21. **public** **void** setDescription(String description) {
22. **this**.description = description;
23. }
24. **public** **void** setInhabitants(**int** inhabitants) {
25. **this**.inhabitants = inhabitants;
26. }
27. **public** **int** getInhabitants() {
28. **return** **this**.inhabitants;
29. }
30. **public** **void** changeInhabitants(**int** i) {
31. **this**.inhabitants = inhabitants + i;
32. }
33. Location(String description, **int** inhabitants, State state) {
34. setDescription(description);
35. setInhabitants(inhabitants);
36. setState(state);
37. }
39. @Override
40. **public** String toString() {
41. **return** **this**.description;
42. }
43. }

State.java

1. **package** moumie;
3. **public** **enum** State {
4. EXISTS("Существует"),
5. IS\_WET("Влажный"),
6. HELD\_BY\_OWNER("В руках у владельца"),
7. ACTIVE("Ожил"),
8. COLD("Холодный"),
9. FOLDED("Сложенный"),
10. UNFOLDED("Расстелена"),
11. BLOWING("Дует"),
12. HURRY("Суетится");
14. **private** String description;
16. **public** **void** setDescription(String description) {
17. **this**.description = description;
18. }
20. State(String description) {
21. setDescription(description);
22. }
24. @Override
25. **public** String toString() {
26. **return** **this**.description;
27. }
28. }

StoryNotFinishedException.java

1. **package** moumie;
3. **public** **final** **class** StoryNotFinishedException **extends** RuntimeException {
4. **public** StoryNotFinishedException(String errorMessage, Throwable err) {
5. **super**(errorMessage, err);
6. }
7. }

NotEnoughItemsException.java

1. **package** moumie;
3. **public** **final** **class** NotEnoughItemsException **extends** Exception {
4. String message;
5. **public** NotEnoughItemsException(String description) {
6. message = description;
7. }
8. @Override
9. **public** String getMessage() {
10. **return** message;
11. }
12. }

BoatDestructionException.java

1. **package** moumie;
3. **public** **final** **class** BoatDestructionException **extends** Exception {
4. String message;
5. **public** BoatDestructionException(String description) {
6. message = description;
7. }
8. @Override
9. **public** String getMessage() {
10. **return** message;
11. }
12. }

ParentTest.java

1. **package** moumie;
2. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
4. **import** org.junit.jupiter.api.DisplayName;
5. **import** org.junit.jupiter.api.Test;
6. **import** org.junit.jupiter.params.ParameterizedTest;
7. **import** org.junit.jupiter.params.provider.CsvSource;
9. **public** **class** ParentTest {
10. **private** Parent parent;
11. @Test
12. **public** **void** initiateTest() {
13. parent = **new** Parent("Мать", State.EXISTS, Location.BEACH, "Ж");
14. assertEquals(parent.getName(), "Мать");
15. assertEquals(parent.getState(), State.EXISTS);
16. assertEquals(parent.getLocation(), Location.BEACH);
17. assertEquals(parent.getGender(), "Ж");
18. }
19. @Test
20. **public** **void** initiateTest2() {
21. parent = **new** Parent("Отец", State.ACTIVE, Location.BOAT, "М");
22. assertEquals(parent.getName(), "Отец");
23. assertEquals(parent.getState(), State.ACTIVE);
24. assertEquals(parent.getLocation(), Location.BOAT);
25. assertEquals(parent.getGender(), "М");
26. }
27. @Test
28. **public** **void** itemCollectionTest() {
29. **class** TestRock **extends** Rock {
30. **private** **int** testCount;
31. **private** **int** testItems[] = **new** **int**[testCount];
32. **public** TestRock(**int** testCount) {
33. **this**.testItems = **new** **int**[testCount];
34. **for** (**this**.testCount = 0; **this**.testCount < testCount; **this**.testCount++) {
35. **this**.testItems[**this**.testCount] = 1;
36. }
37. }
38. **public** **int** getCount() {
39. **return** **this**.testCount;
40. }
41. **public** **void** setItem(**int** i, **int** j) {
42. **this**.testItems[i] = j;
43. }
44. **public** **int** getItem(**int** i) {
45. **return** **this**.testItems[i];
46. }
47. **public** TestRock() {
49. }
50. }
51. **class** TestParent **extends** Parent {
52. **public** **void** pickUpRocks(TestRock thing, **int** number) {
53. **if**(getItemCount() == 0)
54. System.out.println(getName() + " подбирает несколько " + thing.getDefinition());
56. **int** i;
57. **for**(i = 0; i < thing.getCount(); i++) {
58. **if**(thing.getItem(i) == 1) {
59. thing.setItem(i, 0);
60. changeItemCount(1);
61. }
62. **if**(getItemCount() == number) {
63. **break**;
64. }
65. }
67. thing.setOwner(**this**);
68. }
69. }
70. TestRock rock = **new** TestRock(10);
71. assertEquals(rock.getItem(9), 1);
72. TestParent parent1 = **new** TestParent();
73. parent1.pickUpRocks(rock, 5);
74. assertEquals(parent1.getItemCount(), 5);
75. parent1.changeItemCount(1);
76. assertEquals(parent1.getItemCount(), 6);
77. parent1.changeItemCount(-1);
78. assertEquals(parent1.getItemCount(), 5);
79. parent1.pickUp(**new** TestRock());
80. assertEquals(parent1.getItemCount(), 5);
81. }
82. @Test
83. **public** **void** purposeTest() {
84. parent = **new** Parent("Мать", State.EXISTS, Location.BEACH, "Ж");
85. parent.setPurpose("test");
86. assertEquals(parent.getPurpose(), "test");
87. }
88. }

ChildTest.java

1. **package** moumie;
2. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
3. **import** **static** org.junit.jupiter.api.Assertions.assertTrue;
4. **import** **static** org.junit.jupiter.api.Assertions.assertFalse;
6. **import** org.junit.jupiter.api.DisplayName;
7. **import** org.junit.jupiter.api.Test;
8. **import** org.junit.jupiter.params.ParameterizedTest;
9. **import** org.junit.jupiter.params.provider.CsvSource;
11. **public** **class** ChildTest {
12. **private** Child child;
13. @Test
14. **public** **void** initiateTest() {
15. child = **new** Child("Ребенок", State.EXISTS, Location.BEACH, 10, "Ж");
16. assertEquals(child.getName(), "Ребенок");
17. assertEquals(child.getState(), State.EXISTS);
18. assertEquals(child.getLocation(), Location.BEACH);
19. assertEquals(child.getSkill(), 10);
20. assertEquals(child.getGender(), "Ж");
21. assertTrue(child.isSuccessful());
22. }
23. @Test
24. **public** **void** initiateTest2() {
25. child = **new** Child("Ребенок1", State.ACTIVE, Location.BOAT, 5, "М");
26. assertEquals(child.getName(), "Ребенок1");
27. assertEquals(child.getState(), State.ACTIVE);
28. assertEquals(child.getLocation(), Location.BOAT);
29. assertEquals(child.getSkill(), 5);
30. assertEquals(child.getGender(), "М");
31. assertFalse(child.isSuccessful());
32. child.jumpOut(Location.BEACH);
33. assertEquals(child.getLocation(), Location.BEACH);
34. }
35. }

BoatTest.java

1. **package** moumie;
2. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
4. **import** org.junit.jupiter.api.DisplayName;
5. **import** org.junit.jupiter.api.Test;
6. **import** org.junit.jupiter.params.ParameterizedTest;
7. **import** org.junit.jupiter.params.provider.CsvSource;
9. **public** **class** BoatTest {
10. **private** Boat boat;
11. **private** Boat boat2;
12. **private** Child owner;
13. **private** Child owner2;
14. **private** Sand sand;
15. @Test
16. **public** **void** initiateTest() **throws** BoatDestructionException {
17. owner = **new** Child("Ребенок", State.EXISTS, Location.BEACH, 10, "Ж");
18. owner2 = **new** Child("Ребенок", State.EXISTS, Location.BEACH, 1, "Ж");
19. sand = **new** Sand("Песок", State.IS\_WET, Location.BEACH);
20. boat = **new** Boat("Лодка", State.EXISTS, Location.WATER, owner);
21. boat2 = **new** Boat("Лодка", State.EXISTS, Location.WATER, owner2);
22. boat.bumpInto(sand);
23. assertEquals(boat.getDamage(), 0);
24. boat2.bumpInto(sand);
25. assertEquals(boat2.getDamage(), 2);
26. }
27. }

BDExceptionTest.java

1. **package** moumie;
2. **import** **static** org.junit.jupiter.api.Assertions.assertThrows;
3. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
5. **import** org.junit.jupiter.api.DisplayName;
6. **import** org.junit.jupiter.api.Test;
7. **import** org.junit.jupiter.params.ParameterizedTest;
8. **import** org.junit.jupiter.params.provider.CsvSource;
10. **public** **class** BDExceptionTest {
11. @Test
12. **public** **void** initiateTest() {
13. Child child = **new** Child("Ребенок", State.EXISTS, Location.BEACH, 1, "Ж");
14. Shallow sh = **new** Shallow("Отмель");
15. assertThrows(BoatDestructionException.**class**, () -> { child.maneuverAround(sh, **new** Destructible () {
16. **private** **int** damage;
17. **public** **void** increaseDamage(**int** i) {
18. **this**.damage = **this**.damage + i;
19. }
20. **public** **int** getDamage() {
21. **return** **this**.damage;
22. }
23. });
24. });
25. }
26. }

NEIExceptionTest,java

1. **package** moumie;
2. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
3. **import** **static** org.junit.jupiter.api.Assertions.assertThrows;
5. **import** org.junit.jupiter.api.DisplayName;
6. **import** org.junit.jupiter.api.Test;
7. **import** org.junit.jupiter.params.ParameterizedTest;
8. **import** org.junit.jupiter.params.provider.CsvSource;
10. **public** **class** NEIExceptionTest {
11. @Test
12. **public** **void** initiateTest() **throws** NotEnoughItemsException {
13. Parent parent = **new** Parent("Отец", State.ACTIVE, Location.BOAT, "М");
14. Rock rocks = **new** Rock("Камень", State.EXISTS, Location.BEACH, 10);
15. parent.pickUpRocks(rocks, 5);
16. assertThrows(NotEnoughItemsException.**class**, () -> { parent.createFireplace(); });
17. parent.pickUpRocks(rocks, 1);
18. parent.createFireplace();
19. assertEquals(parent.getItemCount(), 0);
20. }
21. }

Build.gradle

1. apply plugin: 'java'
2. compileJava {
3. sourceCompatibility = '1.8'
4. targetCompatibility = '1.8'
5. }
6. repositories {
7. mavenCentral()
8. }
10. dependencies {
11. testImplementation(platform('org.junit:junit-bom:5.7.0'))
12. testImplementation('org.junit.jupiter:junit-jupiter')
13. }
15. test {
16. useJUnitPlatform()
17. testLogging {
18. events "passed", "skipped", "failed"
19. }
20. }
21. jar {
22. baseName = 'gs-gradle'
23. version =  '0.1.0'
24. }
25. wrapper {
26. gradleVersion = '6.6.1'
27. }
28. compileJava {
29. options.encoding = 'UTF-8'
30. }
31. tasks.withType(JavaCompile) {
32. options.encoding = "UTF-8"
33. }
34. System.setOut(**new** PrintStream(System.out, **true**, "UTF-8"))
35. apply plugin: 'application'
36. application {
37. applicationDefaultJvmArgs = ['-Dfile.encoding=cp866']
38. }
39. mainClassName = 'moumie.Main'
40. task fatJar(type: Jar) {
41. baseName = project.name + '-all'
42. manifest {
43. attributes 'Main-Class': 'moumie.Main'
44. }
45. from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
46. }
47. with jar
48. exclude 'META-INF/\*.RSA', 'META-INF/\*.SF', 'META-INF/\*.DSA'
49. }

**Результат работы программы:**

Тень скалы покрывает Песок

Все посмотрели За борт лодки

Муми-тролль изящно обошел Отмель

Носом Лодка мягко уткнулась в Песок

Муми-тролль держит Фалинь

Муми-тролль выпрыгнул из Лодка на Пляж

Муми-мама перешел из Лодка в Пляж

Берег ожил

Обитатели берега суетятся

Муми-мама подбирает несколько Камень

Муми-мама сложила очаг

Муми-мама хочет подогреть оладьи

Муми-мама подбирает Дрова

Муми-мама поместила Скатерть на Песок

Скатерть Расстелена

Муми-мама поместила Камень на Скатерть

Муми-мама поместила Камень на Скатерть

Муми-мама поместила Камень на Скатерть

Муми-мама поместила Камень на Скатерть

Муми-мама поместила Чашки на Скатерть

Муми-мама закопала Банка с маслом в Песок

Муми-мама подбирает Лилии

Муми-мама поместила Лилии на Стол

**Вывод:**

При выполнении данной работы я использовал в своем коде вложенные, локальные и анонимные классы, конструкции try-catch для обработки исключений, фреймворк Junit для ознакомления с основами юнит-тестирования.